UNIVERSITY OF OSLO
Department of Informatics

Game of Exams

Creating a gamified learning platform to teach university curriculum in programming

Master thesis
(60 credits)

Erlend Larsen
Vestad

August 2, 2013

Abstract

Gamification has become a popular trend to create more engagement in ICT systems in later years. A gamified web platform is created to increase engagement in learning tasks related to university courses in programming. It explores how to design an artifact that applies game-driven structures and incentives to learning. This is evaluated and developed using methods from action design research.

Psychological theories are explored to understand the motivating forces behind meaningful gamification. Through an iterative development process, both technical and theoretical aspects are analysed in the of teaching programming.

The result is webapp which supports creation of interactive java exercises. Students can solve the programming exercises using an in-browser editor. The solutions are submitted to a webservice that is able to run the programs and get resulting output. Students then receive automatic feedback on their solution.

The thesis will discuss the challenges related to creating a platform to teach programming, while using game design methods to create a meaningful learning experience.

Acknowledgements

This thesis has been written as part of my masters degree in computer science, at the Department of Informatics, UIO. I want to thank the people that has supported me during this challenging period. Great thanks goes to my supervisor Margunn Aanestad for your guidance and support. Thank you for your encouragement, and believing in me and the project.

I also want to thank the teaching assistants, lecturers and students who used and participated in Game of Exams. A special thanks goes out to the lecturer of INF1010, Stein Michael Storleer, for his help and cooperation during the testing of Game of Exams in the biggest programming course at the Department of Informatics.

Thanks to my fellow students for good times at the lunch table, especially Martin Sommervold and Eirik Andreas Husabø for their smart wits and good company during long study sessions.

Last but not least, my biggest support, Kristin Solgård. Your patience and understanding is amazing, the best part of finishing my thesis is that I can spend more time with you.

Contents

List of Figures

Chapter 1

Introduction

In the article "Fragmented Future" DiNucci (1999) first coined the term Web 2.0 and wrote:

> "The web we now know, which loads into a browser window in essentially static screenfulls, is only an embryo of the Web to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just starting to see how that embryo might develop"

14 years later we have witnessed how that embryo grew into the the amazing dynamic Web we now know. Together with other technologies born in web 2.0's wake like cloud computing and Google's V8 Javascript Engine, the limits to what a browser can do seem endless. Modern web technology has made it possible for any web developer to create dynamic web sites where users create, edit and collaborate together in dynamic environments. Tasks that previously required expensive personal software, like word processing, has now merged into the browser. Powered by todays web technology the tools themselves evolve to support new activities. One such example is the way you can write documents in Google Drive that supports simultaneous editing of a document online, by multiple users.

This thesis seeks to explore how modern web technology can be used to better education. Specifically the area of teaching and learning programming. Outside the scope of higher education, this area is experiencing an emergence of new tools that facilitate learning and teaching of programming in new ways. The last few years, web services has been created, that incorporate all the latest internet trends and technology, into tools that teach programming. Using one of these websites, you can, from the comfort of your own browser, learn the basics of a new programming language in a matter of days.

Learning the basics of Java at the University of Oslo (UIO), requires one semester of study. As a metaphor to DiNucci's words in the start of this chapter, Education 2.0 has been born, and higher education needs

to get on board. The theme for this thesis is how a tool can be created that incorporates the benefits of modern web technology and trends into higher education. This is done in the context of programming courses at the Department of Informatics at UIO.

1.1 Motivation and background

I have been interested in the idea of gamifying normal activities for years. It was then a pleasant surprise when innovative websites surfaced that utilized the potential of gamification to enhance activites that I do every week. I discovered sites like Code Academy and Code School in the start of 2012, and was instantly amazed at how addictive it made learning a new programming language. Another example is physical exercise, which is another interest I spend much of my time doing. I have been motivated to exercise all my life without any influence gamification, but through the form of a social networking site called "Fitocracy", the experience has been lifted and I have gained even more motivation. This website has gained over a million users in less than two years, much thanks to the gamification aspects of the site. The motivation to jog one extra mile is increased when I know that it will earn me a badge for completing a special milestone like "jog 10 miles in a month". When this is achieved, the badge is displayed at my public profile, possibly leading to feelings of social recognition in the Fitocracy community.

I started studying informatics at in 2007, just around the time Web 2.0 exploded. Since then I have taken many of the programming classes available at the Department of Informatics (IFI). Even thogh I am at the end of my education, I have kept the relation to the beginner programming courses through the position of teaching assistant in Java. Each semester I have been part of a large group of teaching assistants, who together with the the course lecturers, teach programming curriculum to first year students. Through this experience I know how hard it can be to teach, and to learn, the abstract art of programming. Many students find it daunting, as programming might require thinking in ways of abstraction that has never been practiced before.

Through the experience of completing exercises and courses at Code Academy I was convinced that there has to be a potential in trying something like this at the university. The last couple of years I have gotten some experience as a web developer, enough to for me to believe it was possible for me to create a gamified website for programming curriculum at the University.

As a part of my bachelor thesis I completed 70 credits in Psychology, where I was specifically interested in the psychology of human motivation. This interest aligns with the enthusiasm for gamification as a tool in web design, which if implemented correctly, adds extra external motivation to activites like programming.

The combination of academic interests, years of experience as a teaching assistant, and interests in programming both as a developer and a teacher, motivated me to create Game of Exams. Through the design and instantiation of this gamified web application, I hope to gain new knowledge about gamification used in an educational setting, what we can learn about the organization on a course level, and how to design a gamified platform specifically suited to the purpose of teaching and learning programming.

1.2 Gamification

“Gamification” is the use of game design elements in non-game contexts. After being defined by Deterding et al. (2011a), it has been accepted and used in most research of gamification I have examined prior to the release of Deterding’s article from 2011. Deterding’s article "From Game Design Elements to Gamefulness: Defining “Gamification" provides a necessary clarification of the term "Gamification" which has been used in later years, both in the IS industry and research, to describe techniques related to game design elements. Too often they have used the term in much broader contexts, making it hard to know whether the authors are talking about creating an actual game, or just using game elements in a non-game context.

It is important to make the distinction of gamification to the concept of “serious games”. Serious games have existed for a long time and has been used in education and work training, with the purpose of educating players in a subject or skill. In serious games the content is incorporated into a game, often a video game of some sort. In contrast, gamifying content, means adding game mechanics like leaderboard, points and badges to something that is *not a game*, so the content itself is still not a game. When trying to gamify programming, this does not mean making the activity of programming into a game. But by adding motivating game elements that are linked to the activity of programming, we can increase user engagement when learning and practising programming, as suggested in the article by (Muntean, 2011). Muntean’s article is not related to the activity of learning programming, but it suggests gamification of an educational course. Gamification can be applied to any subject, as it the content or activity itself is the same.

For example, let’s say you were to gamify the activity of learning children to read. A serious game could try to teach reading through some sort of a video game, trying to mask the activity of practising reading as being part of the game. This could be done by telling an animated story, where the hero in the game comes across a puzzle: To move on in the story the hero needs to solve a puzzle of letters moving across the screen, a voice sounding "Puzzle", and asking the user to click the correct letters to move on in the game. This changes the activity of reading to something

completely different. If you want to gamify reading, in it's simplest form you could give points for reading a whole page in a book, and a gold star for each completed book. The activity of reading is still the same, the content is not a game, but we have added some possibly motivating game elements, to raise engagement in the activity. There is no guarantee that this actually raises engagement.

1.3 Research questions

Inspired by real world applications of gamified web programming websites [1] this thesis will explore if a similar artifact can be created in the context of university programming courses. The content in the existing applications is specially designed to engage users in learning of different kinds of web technology. If such a platform is to be created for university programming, it will be influenced by the situated environment created by students, teachers, course structure and curriculum design.

1.3.1 How can an artifact that applies game-driven structures and incentives to learning of programming be designed?

The thesis will explore research and theory that creates an understanding of the problem through psychological theories related to gamification and motivation in ecucation. This knowledge is later applied through the creation of Game of Exams. The artifact is tested in a large programming course at IFI.

1.3.2 What problems are faced in the creation and implementation of the artifact in the context of university programming courses?

Through the instantiation of Game of Exams, relevant problems is explored in through a research method called *action design research* (Sein et al., 2011). This method leads to exploration of the problem domain in a setting where the artifact is built and concurrently evaluated to create knowledge.

I hope to provide further insight into persuasive design, gamification, and psychological models of motivation used in the educational setting of teaching programming. Readers of this thesis can expect to gain insight in both theoretical and technical implications of a gamified learning platform for university programming.

[1]

- www.codeacademy.com
- www.codeschool.com

Chapter 2

Related research

In this chapter I will present previous research that relates to the theme of this thesis. My purpose is to get an understanding of previous work, that helps me position my own research in the field. Science is a collective effort, therefore I must understand what research has been done before, to ensure that my efforts add something of value to the collection of knowledge.

2.1 Gamification of education

The gamification trend has only been going on for a few years, so that when I was doing my literature view it's clear that the academic field of gamification is still establishing itself. There is much research to be found concerning video games and "serious games", but as I explained in section 1.2 about gamification, it's not necessarily that similar. Gamification shares many aspects with video games, but because it's applied in a non-game context, it's much more important to focus more on the applied context. For this project, gamification is applied in the context of education, and more specifically in the context of programming. Since gamification is a new trend in academics, it's hard to find research that specifically revolve around gamification of programming in education. Therefore I have concentrated on research of gamification generally in the field of education.

2.1.1 Proposed scenarios for gamification in education

Erenli (2012) wants to demonstrate how much time people spend on playing games, and how there is a big potential in using the interest in games, by using gamification in education. He lists an impressive list of facts about games (and by his account, also gamification). Some interesting facts are:

- The average gamer is 37 years

- 97 % of youth play games
- Female gamers represent 42 % of the gamers, and girls under 18 play more games than boys the same age.
- 77 % of american households own games
- 46 % of working germans are playing during working hours

The facts in Erenli's article continues to demonstrate how much games has "invaded" the society, and how smartphones has exposed a much larger audience to games. The audience of games seems to be much broader than what many people might believe, and it's clear that most young people are quite familiar with games. The facts above makes it fair to believe that gamification of education will be able to hit a wide group of students familiar with games.

The rest of the article sketches a riddle game using geocaching [1]. He proposes a game that send students on scavenger hunts coupled with content related to a education. If in example a math course is using the game, students will have to solve math equations or riddles to move on in the game. Erenli suggests that such a game could be used at the start of the semester to facilitate team building skills and collaboration between students.

After the proposed scavenger hunt riddle game, Erenli briefly mentions the mobile game "Zombies, Run!", a game that tries to immerse runners in a zombie infested game where they collect items while running. He states that it would be interesting to evaluate the motivating influence of such a game.

After reading Erenli's article it's clear that he is excited by the potential of gamification in education. He makes a good case of demonstrating the impact games has on society today. Unfortunately I find the scenarios discussed of little relevance to the thesis, as some are not even related to education ("Zombies, Run!"). The scavenger hunt might be a good way to engage students in small learning tasks to solve riddles, but it seems like a platform that is not very suited to solving more complex problems. Especially since you will have students running around outside looking for geocaches.

When I did my literature review this paper was only available as conference proceedings (Erenli, 2012), but an article has later been published in the International Journal of Emerging Technologies in Learning (iJET) (Erenli, 2013). I agree with Erenli's points of the potential of gamification, but the scenarios mentioned are only sketches and does not mention any relevant theory to be used in my thesis. His main goal seems to be inspiring educators to use the gamification trend, and he extends an invitation to other researchers to collaborate on further research.

[1] Geocaching: Navigating by GPS to find containers hidden outdoors

2.1.2 Gamification of an e-learning platform

Muntean (2011) wants to demonstrate the utility and importance of Gamification in an educational environment. They suggest using gamification to raise engagement in an e-learning course, and continue in the article to discuss ways in which it can be achieved. They article mentions common game design techniques that has been used successfully in many contexts, which they argue can be used just as well in an e-learning platform. Their main arguments to using gamification for a web platform is it's potential to increase intrinsic motivation to learning, and overall increased user engagement on the site.

The article is written in hopes of realising the project, but unfortunately I have not found any articles detailing the realisation of the project. But the article still raised some interesting concerns to gamification of education, as well as relevant theory for gamified web design. The main points relating to my project is the introduction of the Fogg Behavior Model (FBM) for persuasive design. The model argues that to generate certain user behaviour there are three principal factors that needs to be considered. These being – motivation, ability and triggers. The article inspired me to use this in my own research, the Fogg Behavior Model (FBM) is explained in chapter 3 on page 11.

As well as suggesting the use of FBM for gamification, the article also points out an important concern when gamifying an activity. When providing users with new reasons for learning, there is a risk of students attributing that their efforts happens because of external goals (in example points and badges), instead of attributing it to the inherent joy of learning. Avoiding this is of great concern, and it's discussed heavily through the understanding of theories specific to this problem in section 3.1.2 on page 15. The article by Muntean (2011) provides good arguments for using gamification for e-learning, but it remains only as a theoretical analysis. As I have found no articles that document an implementation of the proposed gamified e-learning platform.

2.2 Meaningful gamification

When I did my literature review in the fall of 2012 the research around gamification and education was limited. The continued exploration of gamification research led me to a few articles primarily concerned with the importance of context and gamification. Articles by Deterding (2011) and Nicholson (2012) state that context dependent gamification is critical to the success of a gamified application. The two articles theorise meaningful gamification from different perspectives, both of which are equally important in relation to the context of Game of Exams.

Table 1. Summary of design principles for motivational affordance

Motivational Sources and Needs	Design Principles	Some Existing Design Examples
Psychological: Autonomy and the Self	Principle 1. Support autonomy. Principle 2. Promote creation and representation of self-identity.	Desktop skins, cell phone ring tones, online avatars, application toolbar customization.
Cognitive: Competence and Achievement	Principle 3. Design for optimal challenge. Principle 4. Provide timely and positive feedback.	Games and learning systems with various challenge levels and immediate performance feedback.
Social & Psychological: Relatedness	Principle 5. Facilitate human-human interaction. Principle 6. Represent human social bond.	Group based games (e.g. online Bridge) with a chat section, visualizations of email exchanges over a period of time to show both tasks and social related messages.
Social & Psychological: Leadership and Followership	Principle 7. Facilitate one's desire to influence others. Principle 8. Facilitate one's desire to be influenced by others.	Blogs (satisfy one's desire to influence by authoring, and to be influenced by reading), virtual communities where leaders sometimes emerge.
Emotional: Affect and Emotion	Principle 9. Induce intended emotions via initial exposure to ICT. Principle 10. Induce intended emotions via intensive interaction with ICT.	Slick/attractive look of iPod or cell phones, engaging games, ICT that induce optimal flow experience.

Figure 2.1: Design principles for motivational affordance by Zhang (2008)

2.2.1 Situated motivational affordances

In a discussion of theoretical models to understand gamification, Sebastian Deterding discuss the implications of work on motivational affordances in ICT (Zhang, 2008), and how this research might inspire new theoretical models to understand gamification. Zhangs research approaches ICT study with a somewhat unusual approach, trying to understand user engagement from a motivational standpoint. The result is a set of design principles (figure 2.1) built on a macro theory of human motivation called "self-determination theory". The idea is that people interacting with an ICT system are driven by certain psychological needs, this drives people to pursue activities that saturate these needs. Zhangs design principles are suggested ways designers can make sure that users are motivated to use an ICT system.

Deterding's article states that this is very relevant to gamification as well. But his article is concerned with something he says is a blind spot in the current research revolving motivational affordances and video games research:

> "Their focus is by-and-large limited to the properties of the game artifact, ignoring the impact of the social situation or context in which the artifact is engaged with."
> (Deterding, 2011, p. 2)

Deterding's paper continues to explain how understanding of self-determination theory (SDT) should be used to get a better theoretical understanding of not only the design methods themselves, but the social context that an artifact is engaged with. In fact he calls out that for gamification to be meaningful, it's *critical* that it is supported by the organisational context that forms the social environment which the artifact relates to. Deterding concludes that the paper tries to provide good theoretical starting points for establishing more knowledge around an extended version of Zhang's original motivational affordances. He calls the concept *situated motivational affordances*, which the hopes that further investigation of self-determination theory will help establish as a valid concept in gamification research.

Deterding's theoretical starting points to secure situated gamification is deeply explored in the following theory chapter. Deterding's paper seems to focus on the social context that is created in the organisation around a gamified artifact. Another researcher is also deeply concerned with meaningful gamification that is context related, but he focuses more on the social context created by the user instead of the organisation.

2.2.2 User centered gamification

Nicholson (2012) takes it a step further, and while he recognises Deterding's approach to context dependent gamification. He advocates a theoretical framework more suited to understanding motivationalal affordances from the context that is dependent on the user. He presents research suggesting that what is meaningful to users are highly individual, and presents some theoretical approaches that can be used to facilitate different users needs. By employing the theoretical framework, designers can try to create gamification that is meaningful to as many different kind of users as possible.

Once again, self-determination theory is used to facets a users motivational needs. Even though Nicholson advocates a more user centered approach than Deterding, his research can be seen as a continuation of the conceptual model suggested by Deterding. Nicholson is critical to a design focusing on fitting an organisational context, and is concerned that this in fact can lead to meaningless gamification. By using gamification in way that is designed to fit an organisations goals, we risk trying to create gamified scoring system that are meaningful only to the organisation but not the user. To avoid this the approach needs to be user-centered. Gamification should be designed in a way that focuses on different ways users can achieve outcomes that seem meaningful to them through the game elements. The outcome of the users behaviour should still be coordinate with organisations needs, but the path to that outcome should be as varied as possible to ensure that every user has the chance to identify with the gamified content.

In the search for theory that can be used to create meaningful

gamification, Nicholson lists several promising theories that can help understand it from a user centered perspective. But in order to keep the theoretical framework in the thesis focused and specific, self-determination was chosen to be the most interesting theoretical framework of the ones discussed by Nicholson.

Both Deterding's and Nicholson's research has led to the the main theoretical framework in Game of Exams, being self-determination theory. Through exploration of several sub theories, I will try to establish an understanding of motivational affordances that is dependent to the specific context of Game of exams. During the literature review, many approaches to gamification was explored, trying to find the ones most relevant to education. While gamification of education seems to be a topic that everyone is talking about, it's hard to find examples of applied examples in education, specifically when the context is specifically revolved around gamification of programming courses. This leads to a theory chapter trying to understand how the area of programming courses can be gamified in a way that is both meaningful to the organisation (IFI) and the users (students learning programming).

Chapter 3

Theory

In this chapter I will present theories relevant for gamification in the context of education. The theories presented has been important to understand why existing implementations of gamified learning platforms for programming are successful. And they have been critical in the design of my own instantiation of a an artifact based on the same principles. Later the theory is invaluable to analysing key aspects of the artifacts problem areas. Problem areas that would difficult to analyse without a theoretical framework that highlights less obvious aspects of something that is a technical instantiation. To make a technical solution that supports teaching and learning, the processes of learning must be understood from a non-technical side. Many will think of the field of pedagogy when they think of learning, but it is given little attention in this thesis. The focus is on gamification, and the possibilites in creating an artifact for increased engagement in learning tasks, specifically solving exercises. I hope that studies concerning the quality of learning in gamified learning platforms are done, but for the time being it's important to wait for the academic field concerning gamification and learning is established properly. Because of this I will oversimplify learning of programming, and make the assumption that students who are more engaged in the solving of exercises, learn more.

It is the primary goal of Game of Exams to motivate students to do more programming exercises. Understanding *all* the the motivational forces in play when students make the decision to sit down with a set of programming exercises is impossible. Game of Exams is first a website, but beyond that it's an artifact that influences lecturers, students, teaching assistants, potentially all resources used in a programming course, including course curriculum. This interplay of artifact and organisation will be analysed and discussed later in the thesis. The theories presented is an attempt to get an overview of motivation from different perspectives. There are both internal and external forces that drive our motivation, it is important to understand both from a psychological viewpoint. That is why the theory of Self-Determination is applied. SDT provides tools to understand the interplay between motivational factors, with a well

researched methods theory for understanding both internal and external sources of motivation. After explaining the basic knowledge of SDT, the chapter will focus on applications of SDT relevant to learning, ICT and gamification. After the discussion of SDT theory, the focus shifts more to a theory explicitly developed to explain user behaviour on a website. The theory is called the "Fogg Behaviour Model" (FBM), and is designed to make it easier to understand the behaviour of users of websites. With a mix of applied theory and broad understanding of motivational processes, this chapter seeks to create a strong foundation of knowledge to be used both during the development phase and later analysis.

To understand and influence the behaviour of users, I have primarily used two models, the “Fogg Behavior Model” (FBM) by Fogg (2009), and "design principles for motivational affordance" by Zhang (2008). Both are deeply rooted in psychological theories of motivation. Both models provide unique ways to understand the mechanisms of human behaviour, but has different ways to understand very similar concepts. The FBM is the result of a combination of many psychological theories (a complete list is available at the FBM website[1], and seeks specifically to explain user behaviour with the purpose of changing it (pervasive design).

Motivational affordances is on the other hand mostly influenced by Self-determination theory. This theory is Self-Determination Theory (SDT), and was initially developed by Edward L. Deci and Richard M. Ryan in 1985. Since then it has been researched extensively and has been applied to many different contexts like education, sports, work, health care, well-being, and any other area that is concerned with peoples motivation and self-direction. SDT has proven very useful in understanding the motivating factors behind activities related to my thesis, education and games. The list of research on education in light of SDT is extensive[2]. Studying games and motivation provides unique opportunities for stydying intrinsic motivation. One of the creators of SDT, Richard M. Ryan himself has done several studies of games and SDT, see Ryan et al. (2006), Przybylski et al. (2009).

Self-determination theory is based on research that the idea that people have certain psychological needs, and we seek to keep our needs as well satisfied as possible.

> "Human beings seek out (and continue to engage in) activities if these promise (and succeed) to satisfy motivational needs" (Deterding, 2011)

The main motivational needs are *competence, autonomy, and relatedness*. Predictions of human behaviour can be made by analysing situated need satisfactions, within different contexts. Because of this I use SDT as a

[1] http://www.behaviormodel.org/references.html

[2] http://www.selfdeterminationtheory.org/index.php?option=com_sdt&view=SearchPublications&task=domainSearch&domain=11

way to understand different contexts in the problem area of teaching programming. In the context of ICT design, efforts has been done to create design-principles that addresses different aspects of SDT (Zhang, 2008), to afford user motivation for ICT use. The design principles of motivation affordance will be explained in section (REMOVED, not relevant anymore)

3.1 Self-Determination Theory

Self-Determination Theory (SDT), is a macro theory of human motivation developed by Deci and Ryan in 1985. It's essence is explained best in the words of the authors themselves:

> "Human beings can be proactive and engaged or, alternatively, passive and alienated, largely as a function of the social conditions in which they develop and function. Accordingly, research guided by self-determination theory has focused on the social-contextual conditions that facilitate versus forestall the natural processes of self-motivation and healthy psychological development. Specifically, factors have been examined that enhance versus undermine intrinsic motivation, self-regulation, and well-being. The findings have led to the postulate of three innate psychological needs–competence, autonomy, and relatedness– which when satisfied yield enhanced self-motivation and mental health and when thwarted lead to diminished motivation and well-being. Also considered is the significance of these psychological needs and processes within domains such as health care, education, work, sport, religion, and psychotherapy." (Ryan and Deci, 2000a)

Self-determination theory has has helped researchers examine human behaviour and motivation for almost three decades, and has been heavily applied in education research (Deci et al., 1991, 2001, Kremenska, 2007), and later to examine the astonishing motivational pull of video games (Ryan et al., 2006). Even with gamification research still is in It's infancy, Self-determination theory is frequently mentioned in conjunction with academic gamification research. (Muntean, 2011, Deterding, 2011, Nicholson, 2012, Julius and Salo, 2013). Knowing that students seek to saturate three basic psychological needs, gives tremendous analytical flexibility to understand a gamified application. The three needs – competence, autonomy and relatedness –, influences what activities the students engage in, and how well they perform these activities, as well as the intensity and volition of their efforts.

Competence

To experience competence individuals needs to feel that their capabilities are being used in an effective way in interaction with the social environmentRyan2002. Saturating this need requires opportunities where a students skills can be expressed in a way that seem meaningful in the social context. The need to feel competence motivates students to seek challenges that has conditions where their capabilities of learning and creativity are best applied to enhance their skills. A student experiencing competence is more likely to feel that the behaviour is self-directed, which leads to increased performance and persistence in the task (Deci et al., 1996, 2001).

Autonomy

Autonomy is the need to feel that a behaviour originates from our own thoughts (Ryan and Deci, 2002). Actions endorsed by internal values, based on intentions perceived to originate from ones own mind, allows students to experience the feeling of autonomy (Deci and Ryan, 1987). Behaviour that does not feel congruent with ones internal values, might lead behaviour to be less self-determined (Deci et al., 2001).

Relatedness

Relatedness is the need to feel connected to valued others in a social context. Behaviour that makes people feel like they belong in a community and experiences caring for and being cared for by others saturates the need for relatedness (Ryan and Deci, 2002).

There are many needs that motivate humans, but these are the three found to be most important to drive self-determined behaviour (Ryan and Deci, 1985, 2002). A student that is able to saturate these needs in a learning context, is likely to experience self-determined behaviour which makes the student more likely to learn and complete learning tasks (Ryan and Stiller, 1991, Deci et al., 2001, Bachman and Stewart, 2011).

3.1.1 Extrinsic and intrinsic motivation

We want students to feel active and engaged, and hopefully feel like their learning is self-determined. There are different ways we can hope to achieve this, but first we need to understand a a basic distinction between two types of motivation. People can be intrinsically or extrinsically motivated. Intrinsic motivation is defined as doing an activity because of the inherent enjoyment of doing the task itself. Intrinsically motivated behaviour is performed because of an internal satisfaction of doing the activity, without seeking a specific reward or outcome (Ryan and Deci, 2000b). A prime of example of intrinsically motivated behaviour is a

child playing. Children will play for hours only because of the fun in creative, playful interaction with the environment. As we grow older life becomes full of responsibilities, and we can no longer do everything just for the fun of doing it. One will then have to behave in ways that lack internal satisfaction, but is done because we seek an external outcome. Extrinsically motivated behaviour happens when the activity is done to attain a separable outcome rather than for it's inherent enjoyment.

Here is what Ryan and Deci (2000a) has to say about intrinsic motivation:

> "Perhaps no single phenomenon reflects the positive potential of human nature as much as intrinsic motivation, the inherent tendency to seek out novelty and and challenges, to extend and exercise one's capacities, to explore, and to learn".

Students are motivated to learn because of many different things, but research has shown that those who are intrinsically motivated performs much better(Deci et al., 2001). Students having fun while learning will have a much easier time, doing the tasks required to learn the material. Unfortunately there are big personal differences in what some students think is fun, and some don't. Students who learn things they are not intrinsically motivated to learn, is motivated by external factors like deadlines, grades, pressure from peers, or because they hope to get a good job. Hopefully educators try make the education fun, but unfortunately it's impossible to make learning intrinsically motivating for everyone. Therefore educators rely on deadlines, mandatory assignments, exam qualifications, and other tools that controls students behaviour.

3.1.2 Cognitive-evaluation theory

Intrinsically motivated students is the gold standard for education. Facilitating intrinsic motivation should be one of the most important focus points of the project. To shed some light on how that can be achieved it's helpful to use a sub theory of SDT, called Cognitive-evaluation theory (CET). The theory explores variability in intrinsic motivation, and explains factors that facilitate versus undermine intrinsic motivation (Ryan and Deci, 1985). Studies has shown that the basic needs of competence and autonomy play specifically important roles in the facilitation or undermining of intrinsic motivation. CET states that events that gives feelings of competence (feedback, rewards etc.) facilitates intrinsic motivation for an action, but only if students feel that the behaviour came from an inner locus of control (self-determined behaviour). In other words, if students feel like their behaviour is not a result of their own self-determination, feelings of competence does not increase intrinsic motivation towards an activity. External events like rewards, evaluations, and deadlines have been shown to decrease perceived self-determination, because a student might attribute the behaviour to be a result of external goals. Goals that are perceived to

be set by others (deadlines, evaluations etc.), gives the feeling that the behaviour did not originate from our own intrinsic joy of doing it (Deci et al., 2001, Ryan and Deci, 2000a).

3.1.3 Organismic integration theory

> Self-regulation is analyzed in terms of self-determination theory using the concepts of intrinsic motivation and the internalization of extrinsic motivation. Laboratory experiments and field studies are reviewed indicating that: (1) intrinsic motivation and fully internalized extrinsic motivation are positively associated with high quality learning and personal adjustment; and (2) maintaining intrinsic motivation and internalizing extrinsic motivation are facilitated by social contexts that allow satisfaction of the basic psychological needs for autonomy, competence, and relatedness. Such contexts are ones that are characterized by the provision of choice, optimal challenge, informational feedback, interpersonal involvement, and acknowledgment of feelings.
> (Deci et al., 1996)

We need to use external goals in education, so how can we still facilitate intrinsic motivation? There is a second subtheory of SDT that has explored this problem, it's called organismic integration theory (OIT). Extrinsic motivation is behaviour that is instrumental, that aims toward outcomes extrinsic to the behaviour itself. OIT claims that there are distinct forms of instrumentality, where extrinsically motivated behaviour can emanate from the self to different degrees. In figure 17 we see that motivation can be divided into a continuum of self-determination, with intrinsic motivation being the prototype of motivation causing fully self-determined behaviour. But on a range of how someone perceive the locus of causality to be external or internal, even extrinsic motivation can be perceived as having internal locus of causality. There are two categories where students will experience somewhat internal or fully internal locus of control. The first is by *identified* regulation, where students are able to understand and identify with the importance of external goals or regulation. This can go even further, and students can *integrate* the goals and regulations with their own values and needs. This leads to very self-determined behaviour, and shares many qualities with intrinsic motivation (Ryan and Deci, 2000a).

What we can learn from this is that when we try influence students to do something, we need to facilitate the integration of extrinsic motivation. Research results examined by Ryan and Deci (2000a) reports that contexts that are autonomy supportive has a much better chance of causing integrated regulation of extrinsic motivation. People must be able to grasp the meaning of external goals and regulations, and sense a feeling of choice, volition, and freedom from thinking a certain way. This allows students to actively transform values into their own.

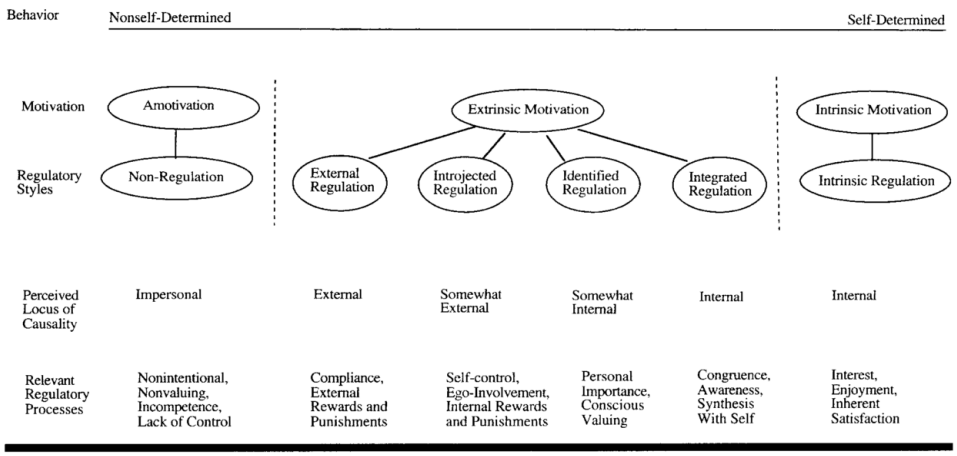

Figure 3.1: The Self-Determination Continuum showing types of motivation with their regulatory style, loci of causality and corresponding processes. Copied from Ryan and Deci (2000a)

Gamifying experiences provide great opportunities for create internally regulated behaviour. Game elements like points and achievement goals represented as badges, is something people are very familiar with. Therefore it's very easy for students to identify with gaols like "get more points", or "obtain the Java lvl 1 badge". They still need to be presented in an autonomy supportive context. Gamification is simultaneously very fragile, in the sense that the moment students feel that the rewards are not meaningful to them, the presence of the gamification itself might feel controlling.

3.2 Fogg Behaviour Model

The Fogg Behaviour Model (FBM) is developed in the Persuasive Technology Lab at Stanford University by Dr. BJ Fogg. Fogg has been trying to understand behaviour in technology use since the 90's. Understanding why some technology works, and some not, is hard. Especially related to human behaviour. After the FBM article came out in April 2009, Google Scholar reports that it has been cited 140 times (7. April, 2013). Fortune Magazine writes in an article from 2008 that Fogg is one of "10 new gurus you should know" in terms of next generation management experts (Web, 12). His research has been applied to improve the products at companies like eBay and Nike (sports technology). Fogg's research was also used in several of the articles examined in my related research section, and is also mentioned in Deterding's articles Muntean (2011), Domínguez et al. (2013), Deterding et al. (2011b), Lockton (2012), Hamari and Koivisto (2013)

Persuasive technology is about learning to automate behaviour change (Fogg, 2009). This is useful to understand, especially when I seek to design web technology that involves a lot of user interaction. As a designer, one

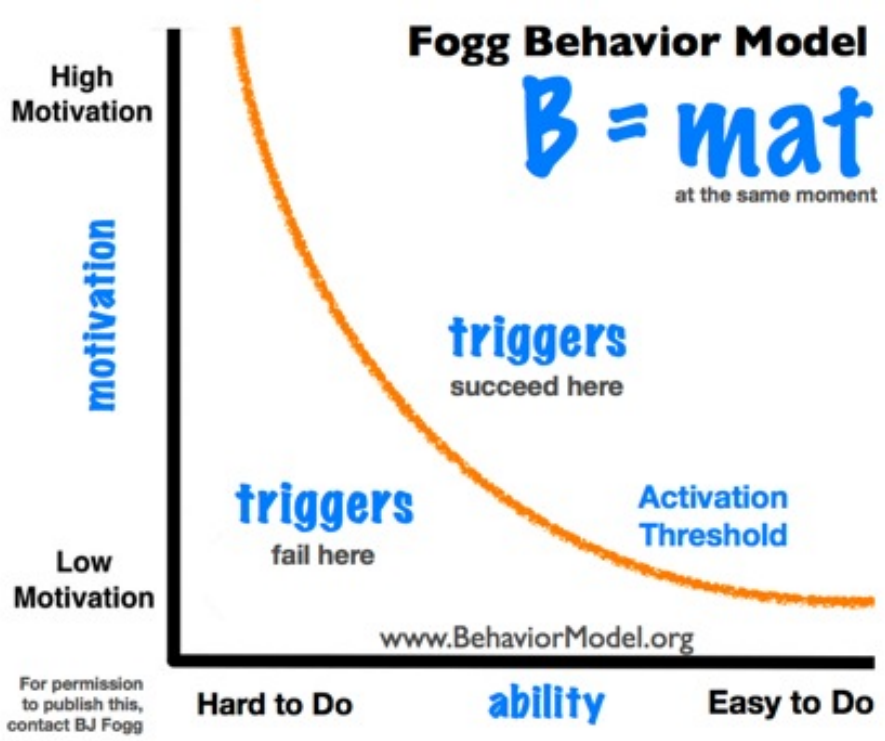

Figure 3.2: Updated visual representation of FBM, from www.behaviormodel.org

often relies on intuition and creativity to motivate users to perform certain activities. Understanding what activities (behaviour) the users should be doing, and then increasing the pervasiveness of the design towards the sought behaviour is key. In our case, that means persuade users to solve programming problems. The article from 2009 describes an understanding of persuasive design through three principal factors: *motivation, ability and triggers*. By understanding the way these factors influence user behaviour, they can be used as measures during development of the artifact, and in later analysis.

The FBM asserts that "for a target behaviour to happen, a person must have sufficient motivation, sufficient ability, and an effective trigger. All three factors must be present at the same instant for the behaviour to occur" (Fogg, 2009, p. 1). The way these affect each other can be visualized as in the following figure:

The left axis represents motivation, which is viewed as being either high or low. The right axis represents ability, with high ability (easy to do) being at most right position. The model is conceptual, so there are no scales, no numeric values. Each factor is either high or low, or somewhere in between. If both ability and motivation is high, you can see that the activity threshold is line goes down, meaning that the chance that the user performs the activity is greater. But even when the activity threshold has been reached, the user still needs a trigger to be “reminded” to do the activity at that

specific moment.

The author has gone to great efforts make the model as simple as possible. It is not limited to understanding behaviour related to design. As an example, the author uses the example of practicing his favourite instrument, the Ukulele, on a daily basis. This is something he is highly motivated to do, and his ability is high (as long as he has the time). But even if he has time and is motivated for daily practice, the activity still needs to be triggered for it to happen. At some point in time he needs to realize that, "this is a good time to practice the Ukulele". As I elaborate on triggers later in this section the importance of timing the trigger will be discussed.

An example more related to web design would be when I read an article on Aftenpostens news website. When I am done reading the article (I have scrolled to the bottom of the page), they have designed an ingenious little trigger that pops up unobtrusively in the right corner of the screen. This box suggests reading an article on a similar topic (trying to match my preferences will likely increase my motivation to read the article), this is an effective trigger to keep me doing the activity they want, reading articles at their website. This is an example of successful persuasive design, as I often keep on reading. There are a few conditions where I will not keep reading, even though the trigger is there. If the article costs money, ability is significantly lowered, because I am usually not willing to pay for internet news articles. But if the article is unusually interesting (increased motivation), to the point where I just *have* to read it, I might be willing to pay afterall. This is an example of a trade off between the two factors motivation and ability. Motivation had to increase for me to cross the activity threshold and perform the behaviour.

To activate a behaviour we need to keep the user in the upper right half of the model, above the activity threshold line, where B (behaviour) = m (motivation) a (ability) t (trigger) are all present at the same time. It is not easy to design for increased motivation. Every user has different interests, so if I am motivated to read an article about, say traveling, another user will finds it uninteresting. Fogg's research suggests that it is oftentimes, easier (costs less resources) to make ability higher by increasing *simplicity* (Fogg, 2009).

One of the greatest examples of this is Amazon's patented 1-click button. They have managed to simplify the most important user activity on their site, purchasing a product, with only one click of a button. Traditional e-commerce websites usually involves many steps in the process of purchasing a product, from putting it in the shopping cart, confirming payment options etc. When the process of purchasing requires less effort, the ability to purchase increases. This allows a trade-off for motivation, where the user might still buy the product, even though motivation was only low or modest. If we look at figure 3.2 on the facing page, the higher the ability, the lower motivation is required to make the user cross the activity threshold, purchasing a product at Amazon.

As these examples show, FBM can be used to explain behaviour, and especially in light of successful pervasive user experiences. As Fogg has analysed design with the behaviour model, a framework has emerged that reveals patterns to understanding each factor. If a target behaviour is not happening, such as using the rating , it might not be obvious if it's motivation, ability or a properly timed trigger that is missing. To find the weak spot in our design, we need to go deeper into the elements of each factor.

3.2.1 Motivation

Not all activities are easy to do (high ability), therefore the user needs increased motivation for the target behaviour to happen. If something is hard to do, like learning programming, we should design it in a way that makes it compelling for the user to put in the extra effort. Specific to this thesis, gamification is the weapon of choice to increase user motivation. Through the project, knowledge of the elements below has guided important design decisions.

Pleasure/Pain

This refers to our instinctive response to stimuli that results in sensation of pleasure or pain. The view of hedonic psychology is that humans are motivated to pursue pleasure and avoid pain (Kahneman et al., 1999, Higgins, 2006). This motivator differs from the others below, that it's a built-in response, and we immediately know if we are experiencing pleasure or pain.

The property of an immediate response, makes it critical for designers to create pleasurable user experiences. The dimension of pain, might not be as relevant, apart from the fact that we usually wish to avoid the user from going through painful user experiences. An example of this motivator in the context of gamification would be pleasant feelings experienced at the moment a reward is received, like getting a badge or seeing your points increase. The sensation of good design is something that is felt instantly, and

Hope/Fear

Closely related to pain and pleasure is the powerful feelings of hope and fear. *"Characterized by the anticipation of an outcome. Hope is anticipation of something good happening. Fear is the anticipation of something bad, often the anticipation of loss"* (Fogg, 2009). Hope and fear explains why we often do things where our pleasure-seeking nature is overridden. One example would be going to the dentist. People will overcome the painful feeling of a root canal treatment, to reduce anticipation (fear) of losing a tooth.

Fogg's opinion is that this is the most ethical and empowering motivator in the FBM, and explains why people join dating sites, or install antivirus software (fear). In education, a powerful motivator is fear of failing on the exam. A specific game element like the leaderboard motivates users by anticipation of climbing to the top, or fear of being bypassed by others and losing your standing.

Social Acceptance/Rejection

The need to feel socially accepted is hardwired into our being, and is the motivation behind much of our behavior. How we behave is undoubtedly influenced by motivation to gain social acceptance, and in many cases avoid terrible feelings of social rejection. Social acceptance influences what clothes we wear, how we act around strangers, and why a teenager just *has* to own the new Apple iPhone.

After web 2.0 it's easy to see this motivator in play, it's the main motivator behind peoples massive interest in social technology like Facebook and Twitter. Many games are interactive social games. Even static content that publicly display achievements, and other personal info are built on our need for social acceptance.

Those with thorough knowledge of psychological research on motivation would notice that FBM has a narrow view on motivation, but the elements above is what Fogg has found to be most relevant to to persuasive design.

3.2.2 Ability

The general term in the FBM is ability, but in terms of design, this often translates to *simplicity*. The reason is that people are resistant to using things that requires effort to learn or use. People are inherently lazy, Fogg says. Instead we should design things so it requires less effort to use, then the likelihood of getting target behaviour increases. With this in mind he has found six common elements that tends to break simplicity. The principle here is that if any of these elements are not simple, the behaviour requires more effort than the user is motivated for.

Time

The user must have sufficient amount of time available, and be willing to invest that time in the target behaviour. Usually people have low patience while doing trivial activities. Consider filling out a user. This is something we are triggered to do quite often, and it's not hard to do. Even the if survey is sent with motivating arguments to the surveys importance, users fail to respond. The reason is simply that it takes time to fill out hundreds of fields, more time than people are willing to give up.

Money

Another resource that varies greatly depending on context, is money. For a student money might be a deciding factor, but for a 50 year old, money requires less effort.

Physical effort

Physical effort sets obvious limits to certain behaviours. An example would be moving a freezer into an apartment in the fourth floor. If I had to carry it through four floors of stairs it would be very hard. But if an elevator were present, the task would be much simpler.

Brain Cycles

This basically means, are we willing to use many brain cycles to do this? Some people enjoy complex thinking, but in everyday actions people usually view them as harder if they require much thinking.

Social Deviance

Breaking the norms of society is usually not viewed as easy. It might require less effort to leave the trash outside the door, but most people will take it to the dumpster to avoid complaints from neighbours.

Non-Routine

People are creatures of habit. Doing the same thing we always do is viewed as easier than breaking the routine. To simplify our life we stick to routines, even though there are alternatives that are time-saving or costs less.

The six elements of simplicity are simple to review as a checklist to find if one breaks simplicity. Only one of these elements needs to be hard to do, to reduce ability. It depends on the context. Paying bills is an activity that is usually the easiest to do right after I get my salary. Other times of the month, my ability to part with money is lower. Fogg describes simplicity as a function of a person's scarcest resource at the moment a behaviour is triggered.

To make the design as simple as possible means finding the barriers to simplicity for our audience. If any of those barriers can be removed, it will increase a person's ability to do the target behaviour. In design it's often easier to make something simpler than designing for increased motivation.

> "People often resist attempts of motivation, but we humans naturally love simplicity" (Fogg, 2009, p. 6).

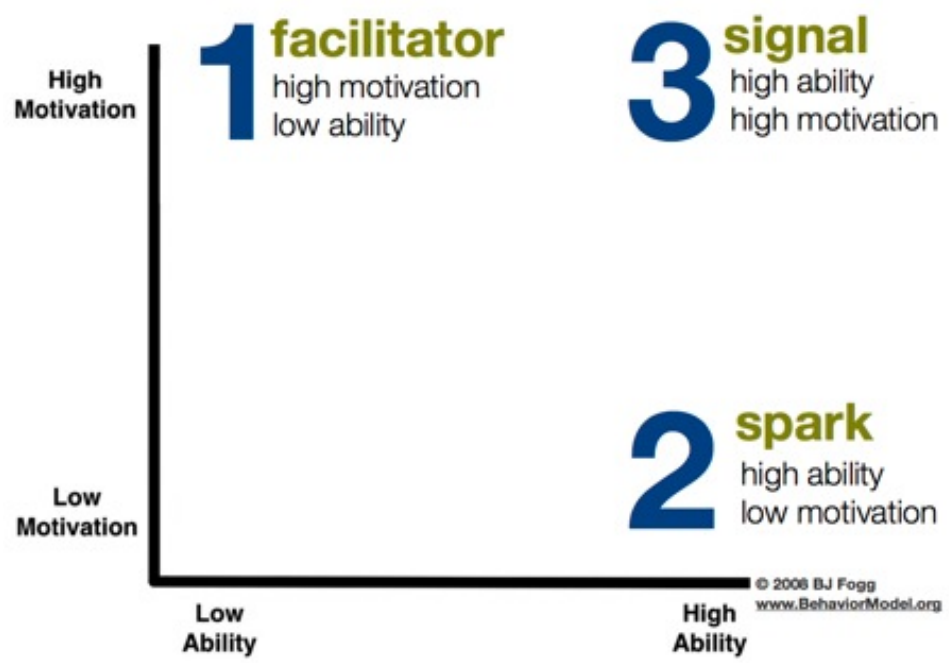

Figure 3.3: The three categories of triggers, to be used in different situations depending motivation or ability levels

3.2.3 Triggers

Triggers are seen all the time in computers, asking you to do things. The screen is filled with programs asking you to install updates, read incoming email, or Facebook reminding you that it's your friends birthday. A trigger is in essence something that tells the user that *now* is a good time to do the behaviour.

Whatever the unique combination of motivation and ability is, a user needs to be guided to perform a behaviour. The way to do this should take into account the situated combination of motivation and ability. Triggers that does not respect this, are annoying and ends up being discarded quickly. FBM divides triggers into three categories, all designed to trigger behaviour, but used under different conditions. If something is hard to do, the behaviour might be *facilitated* to seem easier. When motivation is lacking, the user might need a *spark*. The last category is a *signal*, basically a reminder that works well when the user is both motivated and has ability.

Facilitating trigger

If a user has sufficient motivation, but views the behaviour as hard to do, it can help to facilitate for the users convenience. This can take the form of instructional videos, tutorials, "next step" tips etc. Another example is

the way social networks let's you connect with friends through your email address book. The process of adding many friends at once has been made easy by following a few steps. Ultimately leading you to spend more time being active at their webpage.

The form of a spark or trigger varies greatly, but the important thing is that it is recognised, is associated with a certain behaviour, and is presented when the user has a moment to do the behaviour.

Spark as a trigger

Sometimes the ability is easy to do, but the user needs some motivating piece of information to actually do the target behaviour. This could be motivating text, videos or some other embodiment of the core motivators mentioned earlier. A video could inspire hope, an image could provoke fear, pressing a button could induce pain. An example of this is how programs use error messages that flash with red colours and warning signs, instantly creating a feeling that this needs to be addressed *now*.

Signal

A signal is simply a reminder. These are seen all the time, but are quickly ignored if either motivation or ability is not present. Providing signals is important as users often don't know when it is appropriate to do something. A good example is the purpose of a traffic light. It seeks not to motivate, but serves as an indication that now is the correct time to perform the behaviour.

It is important to understand that the presence of one trigger/motivation element does not rule out other combinations of others being present in the same situation. The framework is only meant as guidelines for effective persuasive design. To exemplify I will make a case of a notice for a dentist appointment. If the recipient has top motivation and ability to visit the dentist, all they need is a simple reminder to visit the dentist. But if motivation is a problem, the notice can contain text that explain the importance of visiting the dentist. To facilitate different aspects of ability one could make the offer appear cheap, if physical effort is a problem, a taxi could be provided, flexible work hours would make timing easier etc.

The concepts of FBM will be further explored as they are applied in chapters 5 and 6 which describe the creation of the artifact. FBM is concidered again in chapter 7 as the project is analysed in light the theories used.

Before those parts are explored I will outline the details of the research framework chosen for my thesis.

Chapter 4

Research framework

The research in my thesis is formed by a discussion of how information systems research is shaped by organisational context during the design process. This thesis specifically want to explore the phenomena of teaching programming by creating an artifact. It's important to ensure that the design process results in some knowledge that can be of use to researchers that want to explore similar concepts. Hevner et al. (2004) wants to secure relevant research results in design research, by combining concepts from both behavioural science and the design-science paradigm. We should not marginalise the importance of organisational context, and by using

> "The behavioural science paradigm seeks to develop and verify theories that explain or predict human or organisational behaviour. The design-science paradigm seeks to extend the boundaries of human and organisational capabilities by creating new and innovative artifacts".

Hevner presents a framework that conceptualises information systems research by combining the paradigms of behavioural science and design-science, this is illustrated in figure 4.1. By researching technological artifacts and how they function and develop in their respective environments, we can gather knowledge to be used both to create truth (behavioural science) and utility (design-science). By using common design evaluation methods to study the artifact, it's implications is assessed in relation to it's environment. Hevner's framework is guided by seven guidelines that helps ensure that the research is relevant and accurate. After evaluations are made, researchers can go back and create new design alternatives, which in turn can be evaluated again. Doing so creates a classic iterative design loop, that tries to find the optimal means to solve an information systems problems.

During development of Game of exams, it has been constantly been shaped by organisational needs. While the purpose is to create something that helps students learn programming, the artifact is supposed to fit with

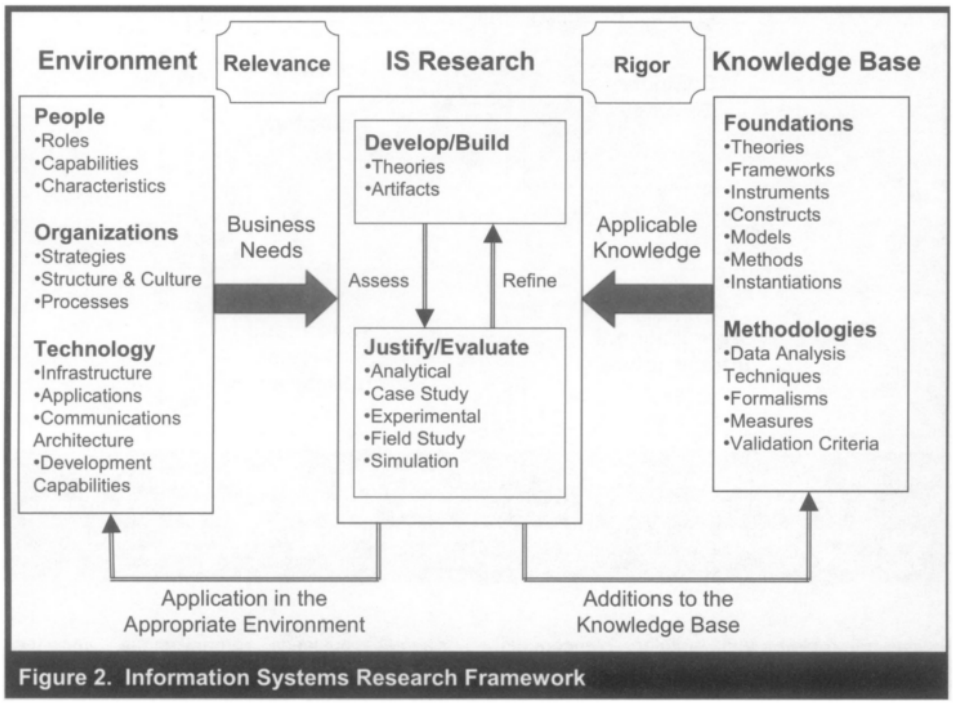

Figure 4.1: Framework for IS research by Hevner et al. (2004)

the university courses that teach programming. This means that both development and use of Game of Exams has been dependent on how IFI organises it's programming courses. The context created by programming courses is embedded in the design, which beyond the technical can be seen as an extension of the organisational context set by course lecturers, administration, teaching assistants and students.

4.1 Action Design Research

The classic create/evaluate loop that is frequent in design research is criticised by Sein et al. (2011) that it does not recognise the way artifacts are simultaneously shaped by the organisational context during development and use. Instead of doing the evaluation after artifact design and creation, he advocates a more concurrent evaluation process. Concurrent evaluation recognises the inseparable activities of artifact building and the organisational context that shapes the process. He creates a new design research method to support his concerns with previous research methods. This is called *action design research*.

> *ADR is a research method for generating prescriptive design knowledge through building and evaluating ensemble IT artifacts in an organisational setting. It deals with two seemingly disparate challenges:*

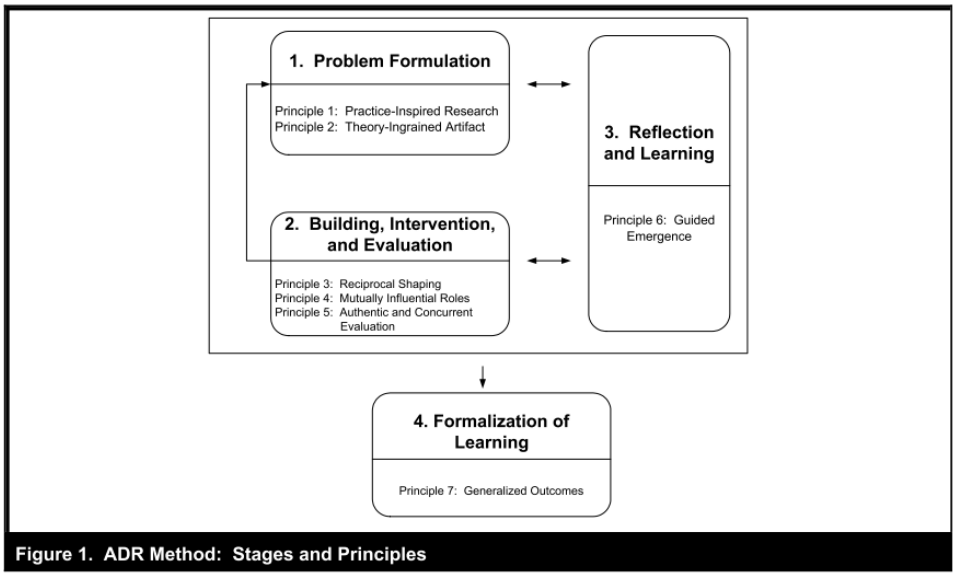

Figure 4.2: Framework for IS research by Hevner et al. (2004)

> *(1) addressing a problem situation encountered in a specific organisational setting by intervening and evaluating; and (2) constructing and evaluating an IT artifact that addresses the class of problems typified by the encountered situation.*
> Sein et al. (2011)

4.2 Stages of research

The way this thesis follows the ADR method is by addressing different stages of the ADR method throughout the thesis. The first stage of ADR is problem formulation, which is done in the first three chapters of the thesis. Research questions is articulated to guide my research efforts, and through the process of a literature review I try to gather knowledge about previous practice. Inspired by earlier research, I gather theory that are relevant to the understanding and development of the artifact.

The premises defined in the three first chapters builds up the second stage of the thesis, which is the building, intervention and evaluation of the artifact. This is described in the project chapters, consisting of an alpha period and a beta period. This process is illustrated in figure 4.3.

Game of Exams is first implemented on a small scale, as a proof of concept. It is developed over a short period of time, where it's implemented and evaluated in a small scale environment, running over a few weeks. In the beta period of the project, it is implemented at a much larger scale, being evaluated in seminar classes taught in the biggest programming course available at the Department of Informatics, UIO. This stage draws

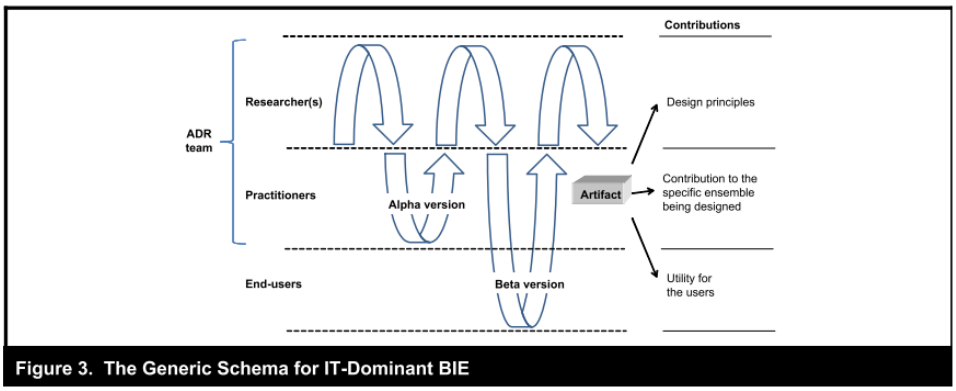

Figure 4.3: Stage 2 of ADR. Illustration from Sein et al. (2011)

on principles that demonstrate the inseparability of artifact building with the organisational context.

Stage 2 of the design process is influenced by reciprocal shaping of both technological design choices, and organisational context. As tools are created to make programming more fun, the tools are constantly formed by the need to fit with course curriculum and the way the course progresses through the semester.

Mutually influential roles shape the project in the that I as a teacher try make the learning platform fit the curriculum and teaching methods in INF1010. I am also the developer trying to choose technological solutions that can solve the problems experienced. Lastly the practices of students and teachers influences the use of Game of Exams, which then shapes further development.

The last force shaping the building stage of the artifact, is the concurrent evaluation done during development. By monitoring the use of Game of Exams both inside and outside of the classroom, authentic evaluation is achieved that assess the artifact in the natural environment. Sein points out that this process might lead to less controlled evaluation than the traditional model of building, then evaluating. The concurrent model supports a model where evaluations happens during the creation of the artifact, where it's natural to take the opportunities of evaluation that arises during development.

The most planned evaluation mechanism was the weekly seminar classes that were useful to observe students interacting with the artifact. In the project chapters, I find other means of evaluation that are opportunities provided after technological choices were made during development. These nature of these evaluation methods is described in those chapters, as it was a natural part of the development process.

In the last stages of the thesis, I analyse the artifact in respect to my theory. The purpose of this is to reflect and learn, hopefully to write down reflections that will help guide researchers that examine a similar problem

domain later. The last stage of the research is formalising the learning, trying to refine earlier research, theories used, and maybe create some new design guidelines for similar problems. This is the weakest part of my thesis, as so much time were spent in the stage of building, intervening and evaluating the artifact. But I hope the concluding chapters of the thesis still provide perspectives that is useful for later research.

Chapter 5

Alpha

This chapter marks the start of the development period, which in action design reserach is called stage 2 of the research. This process is divided into two periods, alpha and beta. Alpha is a small scale proof of concept instantiation of the artifact. This chapter might be surprisingly long compared to the small period that the alpha was tested (three weeks). But this is still the period where the ground work was done in establishing much of the core functionality of the webapp. This chapter will then describe the process of developing the alpha of Game of Exams.

5.1 Preparations

The first period of the project I focused much on finding a technological framework that would allow me to do rapid changes to the functionality of the artifact during the whole project period. During evaluation I would discover new requirements that required rewriting core functionality, therefore the importance of finding a flexible technology stack was very important. I settled for an exciting new framework called Meteor.js.

Meteor.js is designed to make it easier for developers to create rich dynamic webapps, which is exactly what the project needs. Browsers and HTML was initially designed to display only static content, but the last ten years the capabilities of web technology has changed. Browsers can now render much dynamic content, but developing dynamic interfaces often requires a lot of work. Meteor.js facilitates this with a combination of technologies to allow programming to done only in the javascript language, both on the client and the server. Combined with a templating system using Handlebars, HTML5 and CSS, meteor.js tries to bridge the gap between the different technologies required for a dynamic webapp. Meteor.js also has patterns for modular development, which is a critical component in a rapidly changing prototype. Modular development allows certain parts to be changed without it affecting other parts of the webapp.

5.2 Proof of concept

In the fall of 2012 I was employed as a teaching assistant in the course INF3331 at the University of Oslo. The topic of the course is problem solving using high-level languages, especially Python and Bash programming. The difficulty of the course is 3000 level, aimed at bachelor students with previous programming experience. Each week the course released a set of small programming problems, which the students had to solve individually. At the end of the course period, the students had a few weeks without exercises or lectures, to focus entirely on course repetition before the exam in December. My plan was to develop a Proof-of-Concept instantiation of the Gamified Educational Programming Platform and test it during the weeks leading up to the exam. I set up three two-hour classes where students could show up to work on old exam exercises. The plan was to have the artifact ready with old exams published in Game of Exams. This gave me the opportunity to get feedback on the artifact in a natural teaching environment. During the alpha, this was my main method of getting feedback about the artifact.

Development of the prototype started early in November 2012. The goal was to create a web application that supported the most basic requirements for the gamified programming platform. This would serve as a Proof of Concept (POC) application to outline the direction of continuous efforts. A Proof-of-concept instantiation is a good way to reveal difficulties about a project that are hard to identify beforehand, either technical or conceptual. The development of a POC might also show that the project should be abandoned.

The goal for the POC artifact was to let students solve exam exercises in the browser. To make this possible I needed an interface for teachers to make an exam set, and an interface for learners to see the exam and solve it. The next section describes the structures that were developed to support this.

5.3 Represesenting programming exercise sets

Exercises in programming exercise sets are usually a mix of natural language text, and some bits of programming code. Occasionally figures and pictures are used as well. From previous years, INF3331 had three exam sets available, these were the exams given in 2009, 2010 and 2011. They were available at the course site in PDF format. In figure 5.1 on the next page you can see a typical exam exercise in the INF3331 course. My basis for the POC was to At the top there is a small amount of natural language text, followed by a code example. The way this code is presented is not in line with the way programming code is usually displayed. A core functionality of any code editor is to display the code with syntax highlighting.

4: Class programming (7 points)

Consider the following class:

```
class Foo(object):
     def __init__(self, i, j):
          self.i, self.j = i, j

     def __str__(self):
          return "(%d, %d)" % (self.i, self.j)

     def __setitem__(self, idx, v):
          if idx == 0:
               self.i = v
          elif idx == 1:
               self.j = v
          else:
               raise RuntimeError("Index out of bounds [0,1]")
```

Make a subclass of `Foo`, named `Bar`, that implements the special methods `__eq__` and

Figure 5.1: Exercise 4 from the 2010 exam in INF3331 without syntax highlighting, as displayed in the original pdf

5.3.1 Syntax highlighting

This refers to the process of colour coding text and symbols of programming code according to the rules of the programming language. This allows the programmer to easier grasp the meaning of the code. Writing and reading code without syntax highlighting feels unnatural for any programmer. Most web services that display lots of programming code understands this, and displays code with syntax highlighting (i.e. Github[1] or StackOverflow[2]).

Implementing this became a priority early on in my project, and was solved using the library highlight.js. The end result is shown in figure 34. Adding this functionality allows teachers to create exercises in Game of Exams that allows programming examples to be visually recognised and understood as fast as possible. The FBM rationale behind this is increasing ability. Clear programming examples will make the learner understand coding examples better.

5.3.2 Exercise creation interface

The admin interface allows creation of a new *Courses*. In the alpha period, the only course that was added was INF3331, but the platform supports adding an arbitrary number of courses. A course will then have one or

[1] https://github.com/erlendve/hibernate_tutorial/blob/master/src/main/java/org/hibernate/tutorial/EventManager.java

[2] http://stackoverflow.com/questions/6638321/how-to-exit-two-nested-loops?lq=1

Figure 5.2: The exercise from figure 5.1, displayed in Game of Exams with syntax highlighting

more *exercises sets* belonging to that course. An exercise would in INF3331 be an en exam set, but an exercise set can be any group of exercises that relate to each other (like a set of weekly exercises often used in programming courses at IFI). In figure 5.3 a screenshot of this structure is shown.

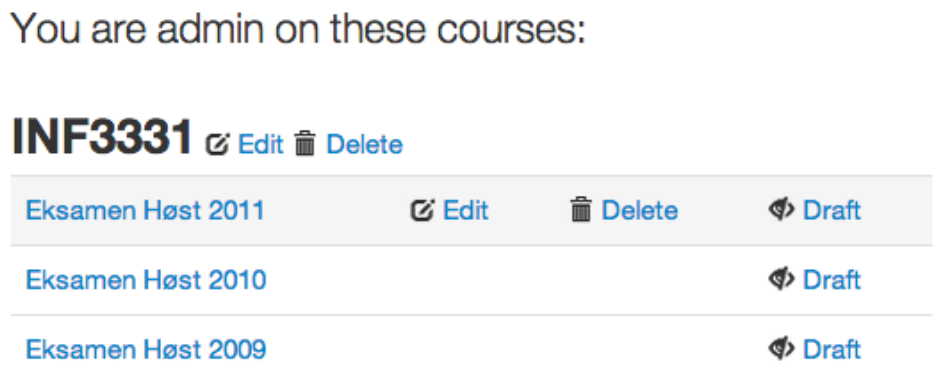

Figure 5.3: Screenshot of the main admin view at goe.meteor.com at first implementation

When it is time to create a new exercise, the administrator is presented with several input fields. The attributes of an exercise was modeled after the information available in the exam sets from INF3331. As you can see from figure 5.1 on the previous page, an exam exercise in INF3331 had a number, a title, points, and the exercise text. These attributes were modeled in the POC, plus an additional "letter" attribute, as it is also common to display exercises with a letter after the number (1a, 1b, 1c etc.). Points is used in programming exams to display the importance of an exercise to

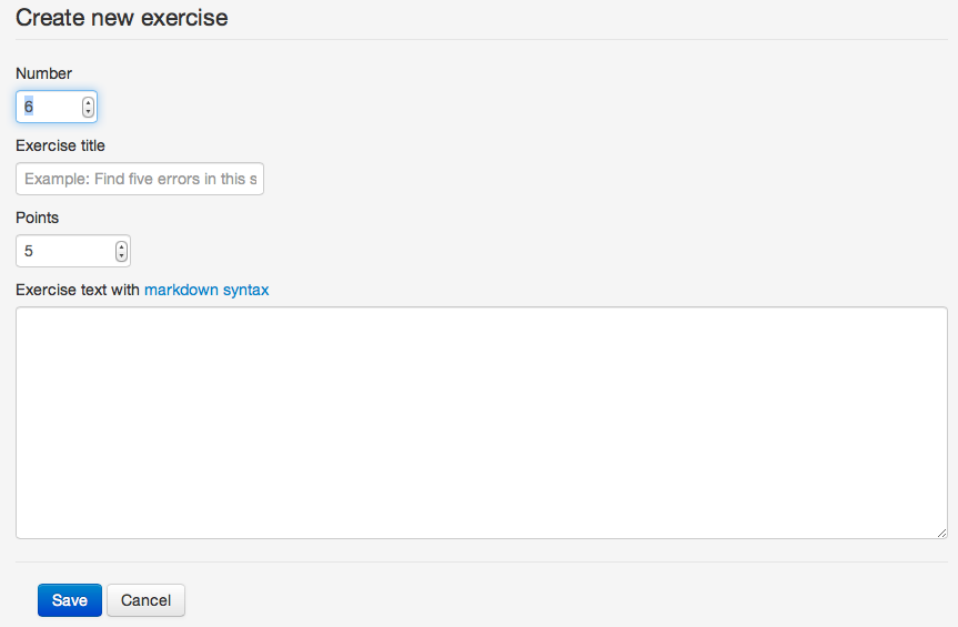

Figure 5.4: Screenshot of the "add exercise" interface available to admins at goe.meteor.com

the final score. An exam could have a total of 100 possible points, and the points of each exercise tells the user what exercises are most important to solve. The add exercise interface is shown in figure 5.4. The end result after completing all exercise fields and pressing the "Save" button is displayed to learners as in figure 5.2 on the preceding page. Changes made to an exam is published in real-time, giving the teacher possibility to add content to the platform at any time. The attribute of an exercise set being either "Draft" or "Published" gives the author of the exercise set the ability to decide whether or not to publish the set to all users. As long as the set is a draft, it is only visible to the owner of the exercise set.

Text to html

A common way of producing exercise sets is using some kind of document preparation system (i.e. Latex), and represent exercises in a .pdf document. The document will often contain natural language text, images and programming code. To make it easy to create new exercises, Game of Exams needed a way to represent the same media in html. But creating an exercise by writing each exercise in html, would take very long time. The solution to this is writing text in markdown[3] format. Markdown is a text-to-HTML conversion tool for web writers. It allows writing in plain text, and converts it to html. Including a markdown processor to the POC technology stack was then a priority. This would allow creating exercises that could use any normal html content tag, supporting headlines, images,

[3]http://daringfireball.net/projects/markdown/

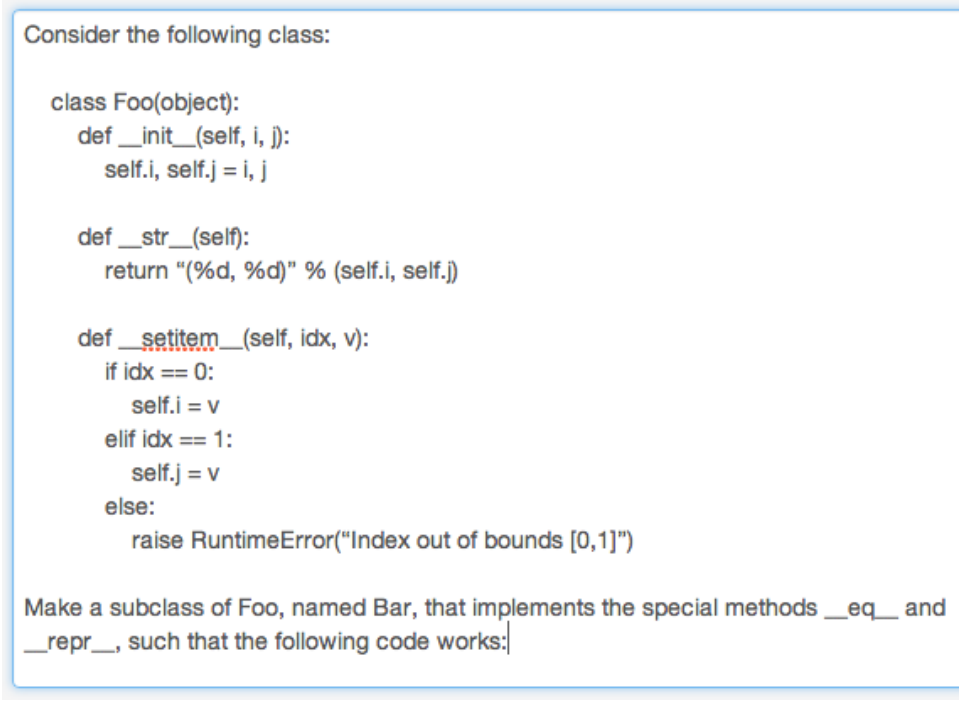

Figure 5.5: Exercise text containing some normal text, and indented code

block lists etc. All frequent tools for structuring and displaying exercise content. Markdown formatting was implemented by converting text input written in the "Exercise text" field shown in figure 5.4 on the preceding page, thereafter processing as markdown, storing the result as html. The result of this conversion, makes it possible to easily create exercises that are displayed as html, with code examples automatically syntax highlighted. In figure 5.5 an example from the exercise creator interface demonstrates how text input looks in a normal html5 textarea. After the text is saved it will be processed to html by markdown. A remarkable feature of markdown makes all indented text automatically to html <code> elements. This allows the syntax highlighter to process the html <code> elements, and the end result is shown in figure 5.2 on page 34.

Later in the project, the flexibility of this input/conversion process is demonstrated by implementing a four page long assignment published on .pdf, rewriting it in the Game of Exams interface to be displayed as html.

5.4 The learner interface

The starting point of the app is an overview of all courses, and their respective exercise sets. These are added to the learner view immediately after they are published by a teacher from the admin interface. In the alpha the interface had only one course (INF3331), with three exercise sets, the exams from 2009, 2010 and 2011. For the first lesson only the 2009 exam

was added, whereas 2010 and 2011 were added for the second class. If one clicked on the 2009 Exam, the view in figure 5.6 was shown.

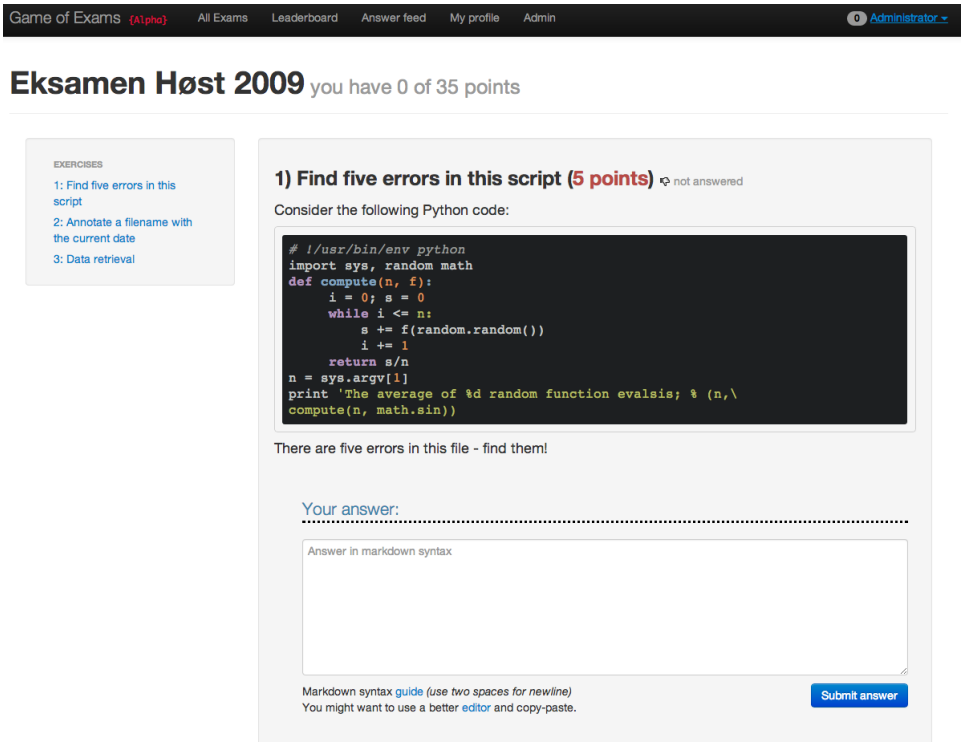

Figure 5.6: The exercise set view for the 2009 exam (version alpha)

As seen in figure 5.6, there is a menu on the left side that displays exercises with numbers, and their titles. At the top is the exercise set title and the progress the user has made in points. The exercise itself fills most of the screen. To navigate between exercises the user can click the left menu, or just scroll down the page and the next exercise will appear. Next to the exercise title there is info about points and some info on whether or not the exercise has been answered. Whenever an exercise is answered three conditions change to notify the learner of the successful progression:

1. The points next to exercise title change from red to green.
2. The exercise is marked with a "thumbs up" icon.
3. Points are added next to the users total points, always visible next to the username in the top right corner.
4. The text displaying total points on the exercise set increases.

To answer the exercise the interface provided a normal html5 textarea. Learners had to use the textarea to answer the exercise, and then click "Submit answer". During the initial design process I had the assumption that markdown format was well known among informatics students, and

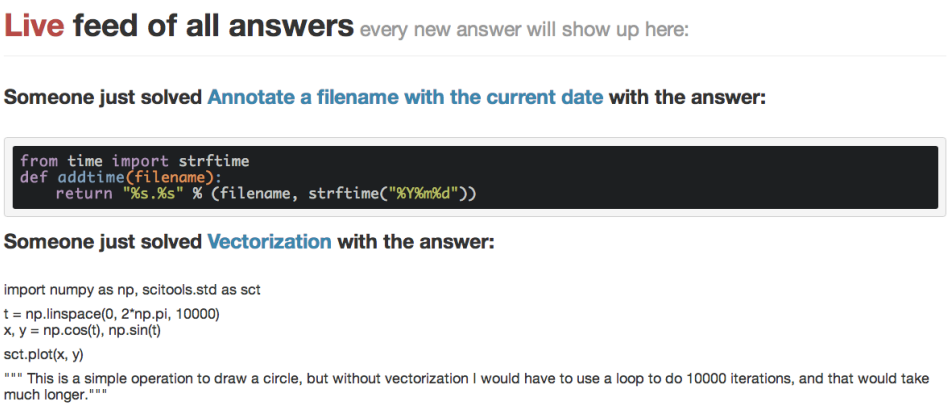

Figure 5.7: Exercises being showed live to all users in the "Answer feed" view (alpha version)

I decided to make the answer interface use this as the default way of formatting the users answers. The interface had text explaining that markdown was being used, providing a link to the markdown guide. After the user had answered the exercise by submitting the answer, the exercise was marked as solved and the right amount of points were awarded.

5.4.1 The Answer feed view

As soon as an answer had been submitted, it would be appear in the "Answer Feed" section of the webapp. This page was shown as in figure 5.7. This was designed based on a couple of reasons: Publically displaying answers makes it possible for users to compare their answer with other users answers. There was another function of this view, which came from the fact that any answer submission would award points. In the alpha, there was nothing checking if an answer was correct, any submitted answer would be accepted and award points. This was an obvious drawback, but in the POC there was no time to create functionality for checking answer correctness. The implementation of the answer feed made all answers visible, hoping that this would make it less attractive to submit unserious answers or gibberish.

As you can see from the figure, answers were anonymised ("Someone just solved...). This was to avoid users feeling fear of social shaming knowing that other users could see their exact answer. The problems of implementing some solution allowing private/puclic sharing was avoided in the POC by just anonymising all answers.

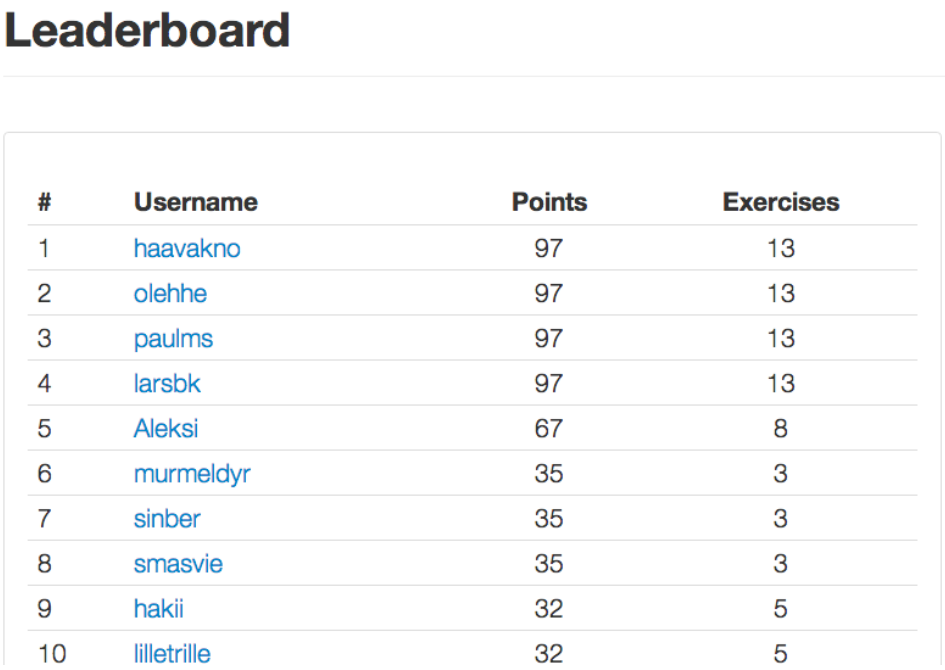

Figure 5.8: Leaderboard displaying all users ranked based on total points

5.5 Gamification implemented in the POC version

5.5.1 Leaderboard

Users receives points by answering exercises. The points rewarded are mirrored to be exactly the same values as in the INF3331 exam sets. In the first week of user testing, the learner interface had only views that displayed exercise sets and the answer feed. For the second week of testing, the "Leaderbaord" view, and the "My Profile" view was added. These are views containing specific gamification content. The leaderboard is a list displaying all users with their points, total exercises done, sorting them based on total points, the user with the most points is at the top of the leaderboard. The leaderboard can be seen in figure 5.8.

The purpose of the leaderboard is to allow users to compare themselves to their peers. The leaderboard is an easily recognizable way of competing against other users, where the goal is to continually improve one's ranking. Whether or not it's meaningful to use a leaderboard is highly situated. In the context of an educational institution course, the users are likely to know at least a subset of the other users. This makes it meaningful to the users, as they can compare themselves to someone they actually know. And gives meaning to the real world. The existing implementations Game of Exams are inspired by does not use leaderbaords. Those sites have a global userbase, where most users don't know the other users, and leaderboards would contain millions of other users. Such a leaderboard would provide little clue to the competence of most users, as the majority of users would

have rankings that put them far down the leaderbard. This would make the leaderboard useless to the majority of the userbase as it is almost impossible to "win" (get to the top). It is also hard to compare yourself to others when there are too many users on the same board, because a small change in points will move a users many rankings above the previous placement, loosing the overview of previous competition.

In the context where all users are participants of specific a institutional course, the leaderboard will be filled with other students the user can relate to, and competing for the top rankings are much easier when there are only 50-300 users. The use of leaderboards is supported in the article by Muntean (2011), based on the way it facilitate status checking between peers.

5.5.2 Badges

The use of badges is not special for games, in the real world it is the same as getting medals in competions, or some other proof of merit. In the world of games they are often represented as achievements, trophys or badges. And usually with a visual icon, in which it can be displayed proudly. The use of badges has become quite popular in recent years, so much it has it's own term: "Badgification". Badges are rewarded for special achievements. In Game of Exams, completing all exercises in a set will reward a special badge. Badges work as a way of helping users setting their goals, as badges work as a quick way of understanding what efforts are worth pursuing, and what those efforts mean. In figure 41 , a subset of the badges available in the POC is shown. As you can see in the figure, there user can also attain badges that does relates to other goals than completing an exercise set. Creating badges that can be attained by completing different kind of goals affords autonomy, as different paths of actions leads toward different rewards. The way the user chooses to attain these rewards is fully up to the user.

Some badges have colour while others are marked in grey. Badges in grey are the ones not yet obtained, while the ones with black text and full colours mark completed achievements. Accompanying a badge is some text that serve as a hint to how the badge is obtained, as well as a fun description of the achievement. The badges in the "Exam achievements" category are obtained solving a complete exam set, and since there are three sets total, there are three badges. Each badge is designed with one, two or three stars, where the ultimate goal is to get the three star badge for doing all exams. In the other category, a few badges are added that are obtained under special conditions. The badge titled "I spam my colleagues" is rewarded for completing an exercise by making a spam-bot that repeatedly send emails to all colleagues every day. If the user did not know about the badge, it will serve as a fun surprise after the completion of this exercise. The badge might also work as a spark to trigger curiosity towards how to obtain this reward. The user might use this curiosity to

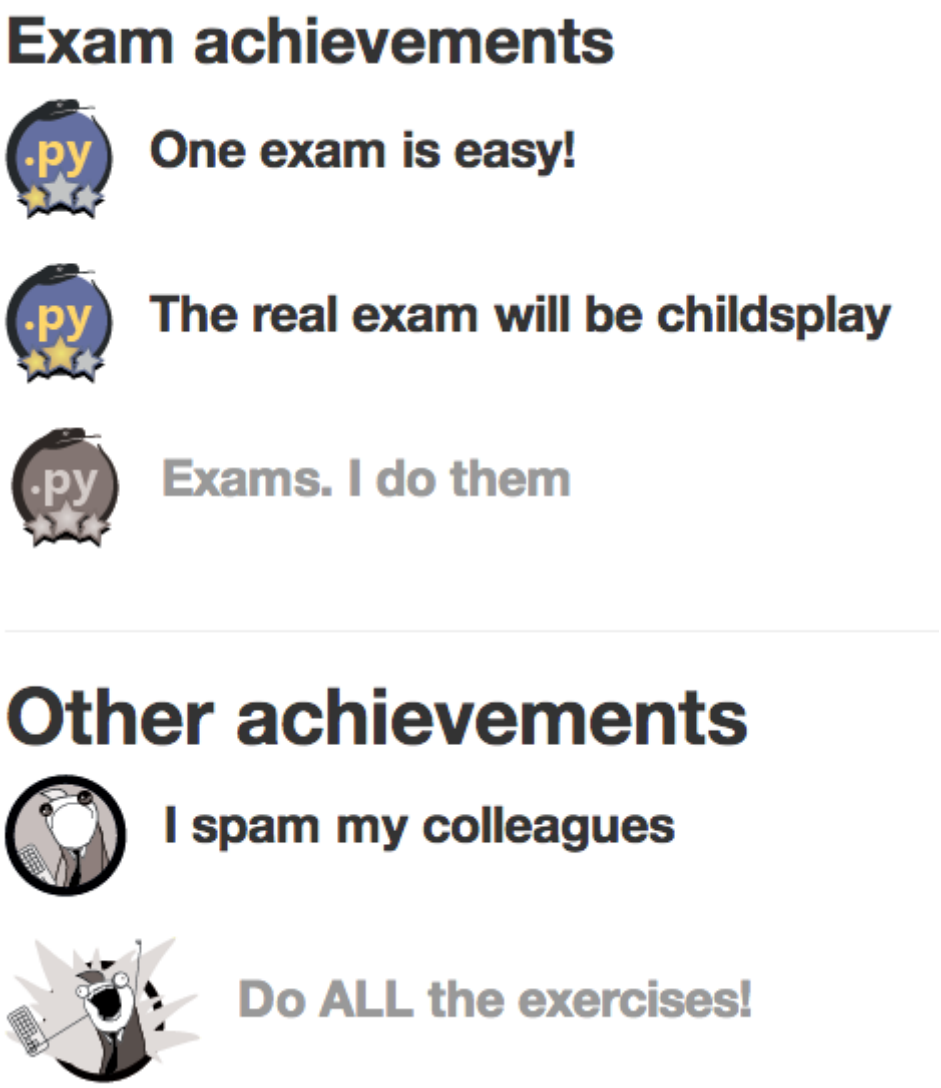

Figure 5.9: Some of the badges awarded for completing certain goals in Game of Exams

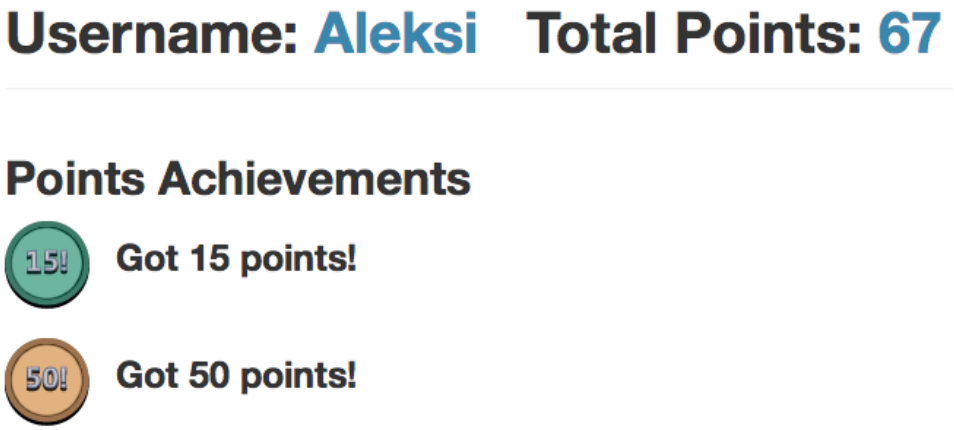

Figure 5.10: A user profile from the alpha version of Game of Exams

look for clues when going through the exam exercises, trying to find hints to how it can be obtained. A user specifically motivated to obtain this badge would probably work harder to complete that specific exercise.

Achievements and badges are publicly displayed in the user profile view. In figure 5.10 we can see the top section of a user profile. Any users profile can viewed by clicking their username in the leaderboard, or by accessing it by an url address (i.e. aplha.gameofexams.com/profile/aleksi). This functionality is yet another way for users to measure their progress against their peers, and is a way displaying status. One's own profile page is quickly accessible under the "My Profile" tab from the top menu. Figure 5.10 also shows another set of achievements, awarded when reaching a set sum of points. This provides yet another way for users to set their own goals, as the next points achievement might not be far away. It is important to balance the amount of badges rewarded, as you usually want the user to have some kind of badge within reach, to make it seem like the next goal is not too far away. Some badges are easy to obtain, while others are harder (like the "ALL exercises" badge"). Getting the hardest badge is in itself viewed as an extraordinary achievement, and getting your hands on that badge might be the motivator for some users to finish absolutely everything. A pitfall of badgification design is to hand out too many badges, making each badge less rare, which diminishes the value of the reward system. The point is to have a selection of badges, where their rarity is clear. In the start we want to make the users "addicted" to the reward system, and there should be some badges that are easy to obtain (like the "15 points") badge. After a while the user might obtain most of the badges, leaving only a few ones left that are hard to obtain. But as the user works his way through the badge system, feelings of commitment might drive the user to continue his/her efforts to obtain the hardest rewards. Knowing that a badge is rare, also provides the possibility of bragging about it to your peers. The status of obtaining these rewards are highly situated, as someone not familiar with the efforts required to obtain it, won't recognise it's

importance.

The combination of the three rewards (points, leaderboard advancement, and badges), provides many ways for the user to decide what goals to aim for next. The use of points in itself might often not count for much, as without any other context the points are almost meaningless. But the accumulation of points leads to more appealing rewards, as it's the way to advance in the leaderboard, and it leads to new badges. The way gamification techniques are tied together, is critical to creating meaningful gamification. The techniques used so far are some of the most basic principles of gamification, other ways of further improving the reward system are discussed in chapter .. ? ..

The rewards facilitates autonomy, competence and relatedness! In terms of FBM, they work as spark triggers, motivating users. They also facilitate ability in the way that the next reward is seldom far away.

5.6 What I learned during evaluation of the alpha version

Repetition classes was set up, and users could choose to attend at their own choosing. The sessions were held in a normal classroom without desktop computers, but I had a projector available for presentations. For the first class, 12 students came. At that time, only the 2009 exam was added to the system. Users logged into Game of Exams by using their own laptop computers, browsing the webpage on the free domain goe.meteor.com. At this point, no other gamification was in place, other than points on exercises. The "answer feed" was also available, and this was continually used to discuss exercise solutions with students.

5.6.1 Problems with answer formatting

As I was watching the answer feed to see how students were doing, it quickly became evident that the students had little previous exposure to markdown format. Even though the answer input field had explicit instructions that answers were saved in markdown format, only few of the answers submitted conformed to the markdown format. This resulted in answers who even if correctly solved, look like pure gibberish after submission. An example is shown in figure 5.11 on the following page, where the user has submitted an answer with programming code, but it displays with an unintended <h1> (heading 1) and all the code joined together in one long line.

In figure 5.12 on the next page a different submission is shown, this conforms to the markdown format, and the answer is evidently much more readable. This answer was submitted by another user, who indented all text in the answer textarea by four whitespaces, which makes markdown

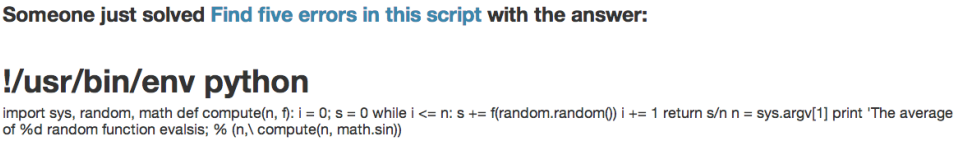

Figure 5.11: Example of an answer that did not follow markdown formatting rules

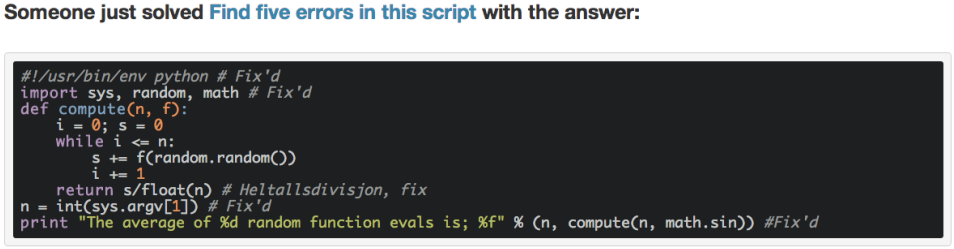

Figure 5.12: Example of a correctly formatted answer

convert it to a html <code> tag. Doing this is not hard, but without this small knowledge, answers are stored as gibberish. This is a very discouraging user experience, and can provoke feelings of pain (leading to less chance of continued usage). Indeed, the feeling of a 15 minutes effort of problem solving, submitting the answer, and seeing it transform into the unreadable code seen in figure 5.11 is not pleasurable.

Based on this feedback it became clear that the app needs clever design to afford easy writing of markdown formatted answers. The default html5 textarea does not work well as a text editor, and it is absolutely horrible to write programming code in. This quickly led to students using their own offline code editor, and copying the code from their own editor into the textfield. From this I identified two needs that had to be addressed in a later version. The artifact needs an online code editor, that affords solving relative small programming problems in the browser. It also needs a text editor tool, that allows text to be written in a format somewhat close to plain text, that converts it to readable html. The process of solving this problem is discussed in section ?.

5.6.2 The need for immediate user feedback

Timely feedback is one of the main design principles of games (Klopfer et al., 2009). This affordance was more or less ignored in the POC version of Game of Exams, as it was clear that implementing feedback would require a whole lot of work. As expected the users commented the lack of feedback during evaluation. They would submit an answer and it would immediately be accepted and points would be rewarded. In essence a user could complete all exercises in the system, answering all exercises with random text. This deficiency undermines the whole value of the reward

system. As users become aware of this, the value of any reward like points or badges are diminished, as they know that efforts to get the reward can be very low. Even the leaderboard becomes meaningless, when you know that the competition might not even submit real answers.

The solution to this problem was the implementation of the answer feed described on page 38. This was a temporary solution to be used in the alpha version, designed to keep the integrity of the reward system intact. If the users realise that any reward can be obtained by minimal efforts, the gamification becomes meaningless. The answer feed plays on users need to avoid social rejection. As answers are displayed publicly, the consequences of ones actions are clear to everyone. Even if submissions are published anonymously, the display of the answer was still public. The intention of the design was that users would perceive the publishing of empty or unserious answers as breaking the social norm. This relied on a shared view among users that the answer feed was important, and filling it with garbage would ruin the experience for everyone. Establishing this norm was facilitated by my influence on students during the repetition classes. The answer feed was frequently used to display user answers on the projector, when I was reviewing the exam exercises at the end of the teaching session. The answer feed worked as a good teaching tool, as it exemplified different ways of solving the same exercise. The use of the answer feed in the class established it's function almost as a public bulletin board to benefit all users. This public understanding seemed to work as a prevention against exploiting the weakness of a missing feedback system. Details of the effectiveness of the answer feed, is revealed in section 5.7 that examines data at the end of the alpha period.

Even though the answer feed temporarily restored the meaning of the gamification system, the need for immediate feedback was clear. In a real exam situation, the user will at some point receive feedback on his/her performance when examination is completed. With a written exam in a programming course at IFI, the student will get the grade weeks after the exam was finished. The grade offers little instructional feedback on how the student can improve future performance. Player feedback is something that is built into all games. The presence of rules that governs what behaviour leads to winning the game is always present. Often the rules are not visible at once, and requires playful interaction with the interface to learn the rules of the game. The mechanism that teaches players these rules are in essence, *feedback*. To gamify a concept, one needs a way to provide feedback to current behaviour. In the alpha version of Game of Exams, no feedback was given when submitting answers. Without a feedback system, Game of Exams is no more than a pretty representation of programming exercises, with the bonus of some easily obtainable rewards. From the start I was aware that feedback was important, but after alpha evaluation, it became a clear requirement for the beta version. The process of finding a solution to the feedback problem is outlined in the next chapter 6 on page 49.

5.7 Results during and after alpha evaluation

As administrator of Game of Exams I had access to the database at all times through the whole evaluation period. This was used to monitor the changes in user activity during the evaluation period. The system was launched at the day of the first repetition class. The launch date was 22.11.2012, and at the end of the first day, the following data was gathered:

- 19 unique users registered.
- 35 unique answers submitted
- One exercise set (2009 exam), with three exercises.

The repetition class had 12 participants, so word probably spread to a few other users during the day. Some students were able to solve the whole exam from 2009 (it had only three exercies) during the two hour class. More content had to be added, to give users more to do. At this point the admin interface had a few bugs, and I was only able to add the 2009 exam before the first repetition class.

During the next week, the admin interface was finished and the 2010 exam added. Before the second repetition class, the system had grown to 31 users, with 51 unique answers. This means that students had been using the system without participating in the repetition class. A message was posted on the course page at 27.11.2012, where the url goe.meteor.com was published to all students. Even though the course page is not checked regularly by all students, this was probably the source of most of the newly signed up users.

At the second and last repetition class, taking place 29.11.2012, there were 9 participants. The "my profile" page was presented to the students. This was the first time students showed real excitement towards the system. The students seemed to quickly recognise the format of the badge system (figure 5.9 and 5.10) , and used words like "cool" and "fun" reacting to the new interface.

After the class at 29.11.2012, students had about a week before the final exam. User activity continued steadily without any more repetition classes being held. The days after all exams were added, a few students seemed to compete to be the first to get "Do ALL the exercises" badge. Two of the students continued to solve all exercises during the next days, continually alternating between first and second place at the leaderboard. At some point one of students took a break, in which the other student advanced to the top of the leaderboard. Both these students were also participants of the repetition class, but during the last week, users who were not participating in the repetition classes started using the system on their own. The day before the exam, a total of four users had answered all exercises in the three exams from 2009, 2010 and 2011. At the day of the exam (05.12.2012), this data was extracted from the database:

- 58 unique users.
- 129 unique answers.
- 3 exercise sets, with a total of 13 exercises.

After the exam, little activity was registered in the system. The course was over, and the main incentive for using Game of Exams was gone (practicing with old exams). A total of 80 students were qualified for the exam, where the number of registered users signals that the project had gained a significant interest among the students taking INF3331.

Of those 129 answers, 19 answers could be categorised as unserious, where the answer was either empty or was in some other way not a serious attempt. 9 of the garbage answers were registered the day before the exam, by the same user. First the user solved four exercises seriously, but then started "cheating", answering the rest of the exercises with meaningless or empty text. By doing this the user got the maximum score and all badges. Through the evaluation period I monitored the answer feed, and if any garbage answers were found I removed them from the database. Considering the fact that any user could choose to exploit the system, I was surprised that it did not happen more frequently. Because of the weakness of missing feedback, the system allowed any kind of answer to pass as correct. The only way to remove "cheating", I had to function as some sort of "moderator", analogous to the role of moderators in online discussion forums. Removing unserious answers was important to keeping the integrity of the answer feed and gamification intact. Moderating with such a small userbase was no problem. But if the system had hundreds of users, it would provide the system with tedious upkeep. The need for moderation would be removed by some kind of answer checking system, which is natural to include in the feedback system planned for the beta version.

Chapter 6

Beta

6.1 Organisational context

In this section I will provide details about the environment the artifact was used in. It is important to understanding the organisational aspects of the artifact. In the alpha version, the organisational context had little impact, but in the beta version the implementation was up and running from the start of the course. The alpha period lasted just a few weeks. In section bloopetibleep I discuss the artifact's interdependence to the organizational context.

The spring semester of 2013 I had secured a position as teaching assistant for the biggest programming course at IFI, INF1010. The course is an intermediate course in object oriented programming using the Java language. The prerequisite for taking the course is a passing grade in one of introductory courses to programming (either Java, or Python). Students who come from the introductory Python course are at a disadvantage compared to the ones coming from the introductory Java course. To ease the transition from Python to Java, a three day course is held to facilitate learning of Java syntax for students who already know Python. Although this works to some extent, it is a known problem that students with a Python background faces a much steeper learning curve in the start of the course. The has around 500+ enrolled students each spring.

The human resources that are assigned to this course are two course lecturers, and a group of 20 teaching assistants. The course has a two hour lecture taught by a course lecturer each week. Additionally there is a three hour long session each week with two experienced teaching assistants that review the weekly exercises with practical examples, doing live coding and real implementations of theory. The rest of the teaching assistants are responsible for students who are divided into groups. Their main work is correcting their students mandatory exercises, while also teaching classes for their group so students can work on the curriculum in a classroom together.

The course material is comprised of slides from lectures, with some additional "notes" collections that are written by the lecturers to go through some of the concepts in a more detailed manner than the slides. Beyond this, students are encouraged to make use of the vast information available about Java and computing, online.

Students have to pass a total of six mandatory exercises to qualify for the exam. A mandatory exercise is submitted through a delivery system, and it's corrected by the teaching assistants in a few weeks time. The workload to pass a mandatory exercise is smaller for the three first exercises, and picks up for the last three. The workload differs heavily from student to student, some use 10 hours to complete one of the larger exercises, while others needs 100 hours. If the student does not pass on their first try, he/she might get a second try if the TA decides that the student made a real effort on the first delivery. For the unfortunate ones who does not pass their first try, this can lead to mandatory exercises piling up. The next mandatory exercise is usually published right after the previous one's deadline. Students who struggle to complete one, will also have a harder time with the next, as they must work on completing the previous before starting the next mandatory exercise.

My role in the course was somewhat different than the other TA's. I got a deal to teach a special class, which would be open to any student of the course. In a two hour seminar each week I would try to set up a teaching program that allowed me to teach relevant curriculum to students, while encouraging them to use Game of Exams. The purpose of this was to get feedback on Game of Exams on a weekly basis, and evaluate how Game of Exams worked as a teaching tool. These weekly seminars were held until mid April, at which point I decided that there was no time to teach seminars while finishing the thesis.

6.2 Developing new functionality

In January 2013 I started working on the beta version of Game of Exams. Even though the alpha version had proved utility of the artifact, it was only a proof of concept instantiation. The requirements for the beta version had to be set much higher, in order for it to be tested in a large course for a whole semester. The alpha version uncovered some critical functionality that had to be implemented for the app to prove real usefulness. This functionality was mainly the need for a *code editor* in the browser, and a *feedback system*, with automatic assessment of exercises. In the alpha version, I had tried to implement a code editor, but the efforts were interrupted as it was too much work for the POV version. The efforts were then picked up again in January.

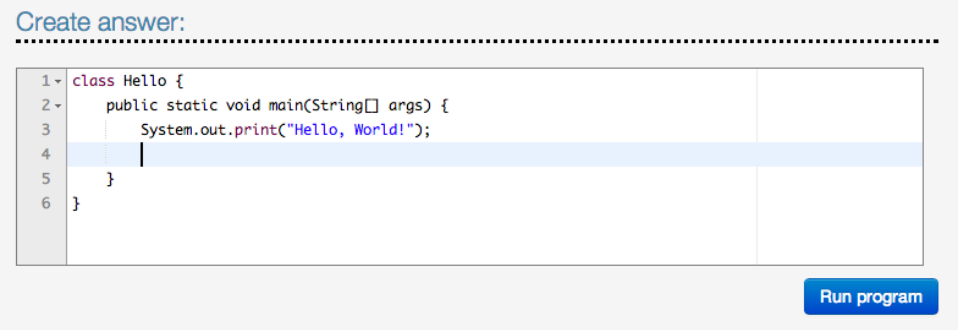

Figure 6.1: First version of the code editor as it was implemented in beta version of Game of Exams

6.2.1 Code editor in the browser

The internet has examples of websites who successfully use an in-browser code editor with advanced functionality as syntax highlighting, automatic indenting, and even code correctness checks on the go. Both Code Academy, Code School, and Khan Academy has implemented in-browser editors for answering their code exercises. After researching the different alternatives out there, I decided to go for the same editor used on Code Academy and Khan Academy. It is an advanced open source code editor called Ace[1]. During development of the POC, I had trouble integrating ace.js with the meteor framework. But with some more time on my hands, the implementation was finished. This opened up many new possibilities for Game of Exams.

The new editor (figure 6.1), replaced the html5 textarea from the alpha versio (shown in figure 5.6 on page 37). The code editor made it more affordable for users to write code when answering programming exercises. The editor behaves as one would expect from many commonly used editors today, with a blinking cursor, syntax highlighting, automatic indents and many more features. It has a wide selection of keyboard shortcuts, who can be set to match keyboard shortcuts from popular editors as emacs or vim. With a proper in-browser editor, users now had a comfortable way to write programming answers. In the alpha version, users would submit their program, it would be stored in the database, and that was it. This is not the common workflow when programming. You want to run your program and see if the code gives the results needed. As long as this could not be done in the browser, users would have to manually copy the program to their computer and run the program. To further facilitate a proper programming environment, the browser would have to run the users program automatically. But how could a browser do this, web browser only understand the scripting language Javascript. As INF1010 is a course in Java, the artifact needed a way to compile and run java programs.

[1] http://ace.ajax.org/

6.2.2 Compile and run

Programs receive input, and return output. Java is a compiled language, where programming code is translated to Java bytecode, which is later translated to machine code by the Java Virtual Machine. Even though today's browsers are very advanced, the only programming language they understand natively is javascript, an interpreted language. Ergo the artifact must execute programs outside of the website itself. That includes other languages than Java too, like Python that was used in INF3331. Building a service that evaluates programs written in many languages would probably be a lot of work, and would be too time consuming to fit in the timeframe of my master thesis. After much research, I found a website that evaluates programs written in more than 40 languages, and prints the result in the browser. It is called Ideone[2] and provides a webservice API, which allows other developers to connect to Ideone through their own programs. After weeks of trial and error, Game of Exams was finally able to evaluate Java programs directly from the browser. To the user, it appears as if the website is evaluating the code, but in fact the code is sent to ideone.com, and the result is displayed in the webapp.

With program execution embedded in the browser, Game of Exams now functions like a simple programming environment. The environment allows coding in a code editor, and execution of the program with the press of a button. The user does not need to install any developer tools on his/her own computer, which dramatically increases simplicity for users. In contrast, for a user to start programming java, the Java Developer Kit (JDK) rneeds to be downloaded and installed (which is not necessarily an easy process for beginner users). Then they need to choose a code editor that also needs to be downloaded and installed. Teaching students how to do this can often be confusing, as every operating system has different tools available. At IFI, students can use programming environment they want, but they have to figure it out on their own. If university computers are used, most of the tools are available on Linux. But today many students bring their own computers, and want to have a working programming environment on their own computer. With Game of Exams in-browser code editor and program evaluation, the students can start coding with just a modern browser. The in-browser interface allows students to write code, submit a program, and receive the program output back. The development period in January ended with some more subtle design improvements, before it was time to start using the artifact in seminar classes.

With program execution implemented, it is also possible to analyze the output to check if the program is correct. A field was added in the admin interface, where one could write the solution output. Whenever an answer were submitted from a user, the users program had to print output matching the solution output set by the exercise creator. This made it possible to check if an exercise answer was correct or not, by checking

[2] www.ideone.com

if output was correct. In the next section, I will write about the results of using the new functionality in Game of Exams in the educational context.

6.3 The evolution of Game of Exams in a real educational setting

This section marks the beginning of an iterative development loop. Major functionality had been added after the alpha version, which changed the possibilities of Game of Exams. But for the artifact to be used in a real educational setting, I had to test it in a real environment. The weekly seminar classes set up an evaluation loop where I would change the artifact each week, then get feedback from the seminar the the next week, and then reiterate. Prior to each weekly seminar class, I would try to prepare relevant content in Game of Exams, using the current functionality supported by the artifact. One of my research questions was to understand if it's possible to use the vast amount of already made content that is being published in the course directly in Game of Exams. That meant that each week I would try to transfer the weekly exercises published on .pdf into Game of Exams.

The first week students were learning Java syntax, so exercises were very basic. But as I was trying to implement the first set of exercises I quickly realised that the current model of creating exercises in the artifact had a very strict form of evaluation to check if the program was correct. In fact, few of the beginner exercises published were worded in a way that even made it possible to check the output. The first exercise was implemented in Game of Exams without trouble, and went as follows:

> "Create a program that prints out "Hello, world!".

Evaluating the output of this program is not hard, as the program has a clear solution: "Hello, world!". But quickly exercises that did not fit the evaluation format followed:

> "Write a program that saves five integers, and then save the average of those in a 'double'."

With this exercise it is impossible to know the final output of the users program. In fact the program asks for no output, and the result is only stored internally in the program in a 'double' variable. How can the artifact facilitate such content? The exercises either has to be specified much more accurately, which looses much of the autonomy in the exercise. If we are to know the output of this exercise we would have change it to using predefined values, and ask for a print of the result.

> Write a program that saves 1, 5, 32, 54, 21 as integers, and then save the average of those in a 'double'. Then print out the double.

By making all values final we secure a predictable output, but the exercise format becomes very limited. In the example above it's also impossible to know if the user just calculates the score manually and prints the solution outright, without following the instructions in the exercise. Good exercises inspire users to think creatively, but a locked format like the one above makes little room for that. When I tried to add the rest of the exercises in the weekly exercises, I saw that almost none had predictable output. For a human it would be very easy to see if the program was correct, but the exercises were made to allow flexible solutions, and it was impossible to know the final output beforehand.

After the first week of seminar classes, it became clear that the evaluation mechanism had to be made much more flexible. This led to a redesign of the admin interface.

6.4 Allowing more flexible program evaluation

The process of adding exercises from the course curriculum into Game of Exams had proven that the current evaluation solution was too strict. Only a very small subset of the course exercises would fit in the format of "Solution output" —> check if user got "Solution output". In the example where the exercise asked for a variable containing the mean, it not make sense to check if the variable contained the right value? Without asking the user to actually print the the double afterwards. This would require us to be inside the program of the user and check the values during runtime.

To do additional checks on the users code, Game of Exams would have to add additional java code to the users program. If the teachers code is in the same program as the users, the teacher can check if the values are correct. If the teacher could paste his own code to the end of the users program, extending it into a java program containing both the code from the user, and the code from the teacher. I hope to keep it clean on the client side, so that users would not know the inner workings of this evaluation process. The student should focus on solving the exercise, and should not need to see the teachers ways of evaluating the code. Therefore the code to evaluate a students program is stored on the server, and then the teachers evaluation checking java code is joined with the users program server side.

To afford more advanced automatic evaluation of answers, I would need a more complex admin interface. I started by adding a code editor to the admin interface, where the teacher could create code that would add to the students answer. The plan was that a teacher could create code that adds to the students program, and then checks if the program is correct.

Challenges with Java

While trying to implement the exercises, I noticed another facet of Java programming exercises that were not supported by the current model.

Many exercises only ask for small programming snippets that does not run as complete programs themselves. The Java language has many rules that make it a hard fit for interactive exercises. Most of the courses offered at Code Academy and Code School are teaching more flexible languages like Javascript or Ruby. Java is a class-based language, and any standard java code must be part of a class. On top of that, any running code must be inside a method that is being called. Normally this is not a big problem when creating large programs. But when we want to make small testable programs it is a big hinder.

Here is an example of a simple programming problem. It is a good exercise, but the answer does not ask for, or need, the extra code required for it to be a complete running program.

> In June it fell 118mm rain, in July it fell 97mm rain, and in August it fell 40% of the total rain for the summer. Create some code that figures out the total amount of rain for the summer, and how much it rained in August. Program it in such a way that if the values of June and July can swapped with new values, while still calculating correct values for August rainfall (always 40 %).

This exercise will likely be answered with only a few lines of code. Which is fine. But Java is a class based, object-oriented language, which requires all running code to be *inside* a method, and even *inside* a class to run. Small Java exercises like the one above are frequent in exams and practice exercises. The one solving the exercise would simply expect to be able to solve the exercise by just providing the solution with the few lines that is actually important to solving the problem. In the context of checking if the answer is correct, this becomes an issue. This being the fact that we need a running program to be able to check if the code is correct or not. I will provide an example answer here:

```
double june = 118;
double july = 97;
double total = (june+july)/0.6;
double august = total*0.4;
```

The exercise is used to show students that math can be expressed in Java, something they can relate to and solve without the knowledge of either class programming or object oriented methods. But to get this program to run we need to add the following code:

```
class RainAugust {
    public static void main (String[] args) {
        double june = 118;
        double july = 97;
        double total = (june+july)/0.6;
        double august = total*0.4;
```

```
        }
}
```

For one thing, the addition of class and method to small programming exercises, are usually not needed nor asked for in the course exercises. But the addition of class and method locks the code inside the scope of that method. This makes it impossible to check the value of the variable (in this case the *august* variable) without being inside the method itself (in this case the *main* method). To make the result of the august variable available for testing, a lot of requirements needs to be added to the exercise, like making the method return the value of the august variable.

The problems faced can be summarised like this:

- Programs are hard to test when variables are in local scope.
- Answers need a lot of extra code to be running programs.

To solve these problems I chose to make it possible to add code "behind the scenes". The goal was to make it possible to answer with code as in the first example. To facilitate this the exercise creator must automatically add the surrounding code to make it run. To make this possible I created two panels in the exercise creator, called "code before" and "code after". In the exercise creator one can then add code here that will be automatically pasted on before the users code, and after. Before submitting the program to ideone.com, the server pastes the users answer together with the code saved in the fields of "code before" and "code after". In figure 6.2 on the facing page I use the new exercise creator to recreate our example exercise in Game of Exams. In figure the new fields in the admin interface adds code that runs the program with code invisible to the user solving the exercise. After creating the exercise in the admin view, the resulting exercise is displayed to the user as in figure 6.3 on the next page.

When the user submits the answer, it will run inside the predefined scope added by the exercise creator. The code now runs, but how do we test if the answer is correct or not? To accomplish this another field named "Tests" was added. Here the exercise creator can add code used to test the exercise. The additional test code will be the last bit of code added to the program before sending the program to ideone.com.

I could have used the "code after" field for adding test code, but during my evaluation classes I saw the need for separating test code from other additional code. The reason for this was that most exercises needed massive amounts of code to tell if an answer was correct or not. While the student is solving the exercise he/she will work in the online editor, altering the program and running it again many times. It's very common to experience compilation errors in this process, and the additional test code clutters the code space for the user. I then made it possible to run the program without adding the test code. While working on the solution the user can run the program without adding the test code, and when the solution is a working program it can be submitted for testing.

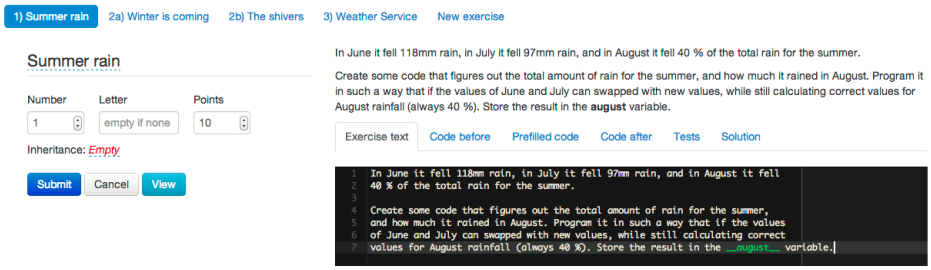

Figure 6.2: The rain exercise is created in the admin interface

Figure 6.3: The rain exercise implemented in Game of Exams, with a code answer written below the exercise text

Figure 6.4: Automatic tests created in the admin interface of Game of Exams

6.4.1 Is the solution correct?

The user seeks to solve the problem, and by submitting the program Game of Exams should evaluate if the programmed solution is correct or not. In the case of our "summer rain" example, the value of the august variable must be checked to determine if the student solved the exercise. As you see from 6.2 on the preceding page the students code is running inside the static method "goeAugust", which is set to return the value of the august variable. I then created test code that checks the returned value of this method. If the value is correct the test class prints out "Tests PASSED", and if the value is wrong the tests print out "Tests FAILED" and a message that explains why the answer is incorrect.

After pressing the "Test" button visible in figure ??, the "code before" and "code after" is added. The program is sent to the Game of Exams server with a flag that tells the server to paste the test code to the program. While doing this the returning program evaluation from ideone.com will be parsed when it is returned to the Game of Exams server. The mechanism to determine if an exercise is solved or not is plain javascript code on the server, checking the output (if the program ran without errors) for at least one occurrence of "PASSED" and no occurrences of the text "FAILED" or "SKIPPED". With this mechanism in place the exercise creator would make sure to use these three words when giving feedback to the user. The reasoning behind the use of these words are explained more deeply in section ?? about the creation of the Game of Exams testing framework. But if the server only checks the output for "PASSED", what if the user creates code in his answer that uses these words? It would be pretty easy to cheat if all the user need to do is to write "System.out.println("Tests PASSED");" in the answer. To avoid this I had to make a mechanism that made sure the

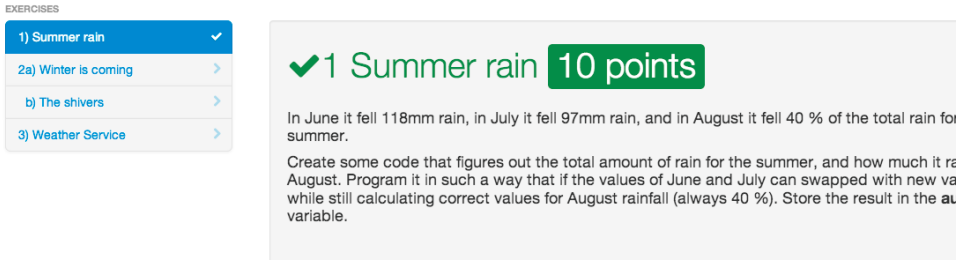

Figure 6.5: The summer rain exercise is solved, and is marked in green with a checked icon

server only checked the output written by the test code, not user code.

The mechanism to secure test output made it a requirement to all exercise tests with the text "_GOE_TESTS_ID", and end with the same text. When the server got a student answer, it would parse the test code before adding it to the program, looking for "_GOE_TESTS_ID". When the text is found (only in the test code), it switches the text with a unique randomised id number, this number is inserted in the test code and stored to be used when the program has been run through ideone.com. After the result comes back, the server makes sure to parse only the part of the output that is printed between the unique id numbers. This mechanism makes sure that the student answer output can not interfere with the output of the automatic tests. If the answer submitted was correct, the tests will print "Tests PASSED" which in turn is parsed by the server. This will mark the exercise as completed, and the student is awarded points, and if the right requirements are met, a new badge. In figure 6.5 the resulting change is shown to the student by adding a checked marker, and the exercise title changes to green, along with a green badge around the points. In the sidebar menu showing all exercises, the exercise is marked as finished by swapping the chevron icon with a checked icon.

After designing the new exercise creator it is obvious why Code Academy and similar sites use scripting languages in their interactive exercises. In languages like Javascript, Python and Ruby code can run without being programmed inside the scope of functions or classes. This makes it much easier to run and test the answers for small exercises like in our Summer Rain example. Even though it is harder to create interactive exercises with Java, that has been my focus as I wanted to examine if it's possible to create interactive programming exercises with university programming curriculum. With the exercise creators possibilities I have tried to create a tool that allows exercises to be created and tested in many languages. Game of Exams allows exercises to be created in many programming languages, and if one would like to create in example Python exercises, it will probably be a lot easier than Java. But I continued my efforts to implement exercises from the curriculum in INF1010, which quickly progressed to advanced topics like data structures. The Summer Rain exercise used earlier is just an example I have used to showcase

different concepts of the admin interface, but the usage has been much more complex than in the example above. In the next section I will elaborate on how a test framework for automatic testing of Java exercises was developed in Game of Exams.

6.4.2 Java testing framework

As I implemented exercises in Game of Exams, I soon realised that the creation of good tests was incredibly time consuming. As the exercises explored advanced concepts, the tests needed to check a vast amount of different cases that could possibly be wrong in each exercise. If tests were not accurate I would risk accepting false answers. Using testing frameworks is common practice in professional software development, and using one like jUnit would be helpful to creating efficient tests. But as Game of Exams used ideone.com to run programs, the only available Java libraries were the standard Java libraries. As I was creating tests for a particularly complex exercise, I began working on a test framework that could be used for testing every exercise in the Game of Exams format. This framework was then made with no dependencies except the standard libraries, so it could be used with ideone.com.

The test framework was developed during creation of automatic tests for the fourth of the mandatory exercises in INF1010. It was one of the bigger exercises in the course, with an exercise text 3,5 pages long on a pdf document. To solve the exercise, students had to create a sorted singly linked list in Java. It would store generic objects, as long as the object implemented the Comparable class. The tests were created as an aid for students during their process of solving the exercise. As a starting point the solution had to be based on an interface that defined the methods available in the data structure. The interface had ten methods, all with more or less clear instructions to their purpose in the exercise text. In the creation of such a data structure, many programming bugs can occur without the programmers knowledge. Testing the solution is necessary to secure a reliable performance of the data structure. Most students has at this point in their education had little training test driven programming. Ready made tests that reveal common programming mistakes in the data structure is an aid both to the students and the programming assistants. The test framework prevents hours of error searching during as the student is working on his/her solution, and is a valuable time saver for teaching assistants as they use the test framework to quickly analyse a students solution for errors while correcting the solutions.

The test framework was built to be reused for creating tests in a consistent way for Java exercises in Game of Exams. It has a few classes and methods that should be used to print test output to users in a format that is easily recognisable both for the users and developers creating tests. A test class is created each time a specific case should be tested. Each instance of a test will be run automatically in the sequence defined by the test creator.

```
------ Failed 2 tests ------
** Testname: leggInn() og hent(int nr) paa indeks - FAILED
Legger inn objekter i kolleksjonen og sjekker at hent(int nr) returnerer
objektene i riktig rekkefoelge

Details: hent(1) returnerer null selv om TestPerson: Bender skal ligge paa
denne plassen

** Testname: tilArray - FAILED
Sjekker om tilArray(V[] a) returnerer alle elementene i samlingen sortert
paa nokkel N.

Details: tilArray(v[] a) returnerer en array hvor indeks 0 i arrayet ==
null, forventet element: TestPerson: Bender

------ Passed 8 tests ------
** Testname: leggInn og inneholder - PASSED - 15 points
** Testname: leggInn og hent paa nokkel - PASSED - 15 points
** Testname: leggInn() og antall() - PASSED - 15 points
** Testname: hentMinste() - PASSED - 15 points
** Testname: hentStorste() - PASSED - 15 points
** Testname: fjernElement() - PASSED - 15 points
** Testname: fjernAlle() - PASSED - 15 points
** Testname: ingen duplikate noekler - PASSED - 15 points

------ Skipped 2 tests ------
** Testname: iterator() hasNext() og next() - SKIPPED
Why: iterator() metoden er ikke implementert
** Testname: iterator() - SKIPPED
Why: iterator() metoden er ikke implementert
```

Figure 6.6: A sample test output during the solution of the fourth mandatory exercise in INF1010

A test must have the following information:

- Title: The name of the test
- Description: A short text describing what the test does
- *FAILED* message (only if the test fails): A detailed message describing why the test failed
- Skipped message: If the test is skipped, usually because other requirements of the solution is not met, the test should be filled with a short message of why the test was skipped.

The Game of Exams java test framework is made to make it easy for developers to create tests when new Java exercises are added to the system. Every exercise added in Game of Exams needs to have precise tests to determine if the student has found a solution to the exercise. As seen in the sample output in 6.6, some tests has failed, and is printed with a detailed message of why. 8 tests are already passed and has awarded 120 points. Two tests has been skipped, as the method is not yet implemented. Along with the purpose of giving feedback to the user, the output is designed to be parsed by the Game of Exams client.

6.4.3 Design implications of the test framework

The creation of a test framework traces back to the need for feedback that was clearly demonstrated during the alpha period of the project. To be able to create feedback, the developer creating exercises must be able to efficiently create automatic tests for exercises. Java is especially in need for such a framework and standardised patterns to facilitate test creation. The design of the test framework was also built with to provide feedback in a way that is encouraging for the recipient. All the other gamified interactive programming sites have compelling ways of giving feedback in a nonintrusive positive way, and some also provide hints. The common pattern is to show a small message of what is wrong with the solution, urging the user to try again. Both Code School and Khan Academy sometimes provide hints on certain exercises. As looking at the hints can prevent the student from independently trying to think of a solution, students are encouraged to not look at the hints until necessary by a clever gamification mechanism. Each hint will subtract 25% of the points awarded for solving the exercise. This goes all the way to the third hint which often directly explains how to solve the exercise, but the student will be left with a 75% decrease in points. The benefit of this mechanism is that it prevents students from abandoning they don't find the solution, while making the price high enough that it's unattractive to check the hints without really trying to solve the exercise first.

In the design of the java test framework, creating proper feedback was a part of the design from the start. Creating tests that determine if a student answer is correct or not is only half of the work. Giving detailed positive feedback to the user is essential to urging the student to continue. As the exercises are completely voluntary, a student can choose to quit at any time. Providing feedback messages that properly explains what is missing, and preferably a message that points the student in the right direction will keep the student engaged. The student is committed to the task of solving the exercise, and feedback is essential to keeping the engagement up.

In practice any test created in the framework starts out with the status of being a successful test. But as the test is runs, the code will try to break the test by seeing of the students solutions has any faults in the code. Whenever a test breaks, a String parameter must be provided with a detailed message of why the test failed. As seen in figure 6.6 on the previous page, the name of the test is printed first, followed by a description of it's purpose. The message starting with *"Details"* is the exact reason why the test failed. The output in figure 6.6 on the preceding page has two failed tests. The first test that fails, checks whether the data structure adds elements correctly, and then uses a method to get an indexed element from the list. For this specific test, there are four different cases that are checked to determine if the methods are working as specified in the exercise text. One way to fail the test is having incorrect sorting of elements in the structure, the tests will then print details explaining that the sorting of list elements are incorrect, provided with an example of correct

sorting compared to sorting in the students solution. But the solution in 6.6 on page 61 fails the test in another way, shown by the *"Details"* text. The student returns a *null* element, where the expected result was the first element added the list. This feedback will hopefully give the student enough information to debug his solution and submit a revised answer.

6.5 Implementing exercises

The knowledge presented in this section details experiences about Game of Exams as as a teaching tool, and how it's affected by the curriculum and what is learnt about the programming curriculum used today. As the evaluation cycles continued on a weekly basis, Game of Exams was slowly formed to match the requirements needed to implement Java exercises published in INF1010. At the start of the Beta period, the goal was to implement every set of exercises that was released every week. The exercise sets was released by the lecturers every, and was available for students to practice relevant curriculum taught the previous week. Doing these exercises were voluntary, and would often be a way for students to learn necessary skills to solve mandatory exercises.

When the first exercises were released for the repetition Java course in the starting weeks of INF1010, I tried to implement some basic exercises using the early version of the exercise creator, but I quickly noticed the limited usage of the automatic testing that was in place. At that point Game of Exams could only analyse textual output from the program, which proved very limited for evaluating most exercises published even with the very basic exercises that was published in the start of the course. The work that followed is detailed in section 6.2. Working to make the platform fit with the Java programming curriculum had a big impact on the design choices taken during development. As I developed the artifact to support Java exercises (in the context of INF1010), I found that it was a unique way to understand the teaching material available in the course. Incorporating INF1010 content in the artifact, then later seeing it in action during seminar classes made me very aware of how the programming problems actually behaved as they were solved. As one is implementing automatic tests for an exercise the creator will have to scrutinise every aspect of the exercise text, and it's possible solutions. This process revealed shortcomings to many exercise texts, or other limitations to their solutions. The lecturers in INF1010 uses a base of exercises from previous years, while mixing in some new exercises every now and then that are recreated to fit current curriculum. There are many human factors that influence this process, such as time pressure, creativity and resources. The ideal situation is to have large amounts of time to create creative programming problems that are engaging and fun to solve. But lecturers seldom have time to sit with this kind of work, and often need to rely on exercise material from earlier years.

It is a known fact between lecturers and teaching assistants at IFI that few students actually do many weekly exercises. This is a problem in most of the programming courses that use large mandatory exercises. The perceived reason is that because the mandatory exercises require so much work, most students don't have time to solve weekly exercises. This behaviour might work to the students disadvantage because solving the large mandatory exercises requires knowledge of the material that was supposed to be practiced in weekly exercises. But during my work with the Game of Exams platform, I got a closer look at the exercises we publish to students. Every exercise implemented in Game of Exams goes through a scrutinous review process as it's being republished in the format of Game of Exams. When implementing an exercise in Game of Exams, the creator will use the admin interface to first write the text in markdown format. This is an opportunity to review the exercise text, which might contain grammatical errors or a confusing message. When authoring an exercise text it's hard for the author to know how the reader will interpret the exercise text. Therefore it is important that the exercise text is reviewed again and again to be as precise as possible. In INF1010 all the mandatory exercises goes through a rigorous review process in the eyes of teaching assistants before being published to the students. Even after such a process it usually takes years before the exercise text reaches the level of clarity we wish to achieve when creating a complex programming problem. Implementing and later evaluating exercises through feedback from students gave a unique possibility to improve the exercises as they were published at the Game of Exams website.

Automatic feedback demands a new level of exercise precision

The pattern for implementing an exercise will first require the text to be written in markdown format. The first step is copying the text from the exercise set, and then doing the necessary modifications to make it into markdown format. This ensures that the exercise text looks good in html format, and provides the benefit of syntax highlighting code snippets (as shown in figure 5.2 on page 34).

After the text is written, the author needs to code the autuomatic tests. This first requires one to create solutions for the problem, and hopefully there will not be too many possible ways to solve the problem. The more different solutions are available, the more tests must be written to secure correct feedback. After doing the process of writing automatic tests, I experienced that to make it possible to create automatic tests, one will often need to add extra constraints to the programming problem. A specific example which was implemented in Game of Exams, was parts of an exercise where students were creating a Sudoku solver. Parts of the problem required a program that could read the contents of a file containing numbers and characters to a sudoku board. This was part of a larger exercise where the goal was to solve a sudoku board with certain fixed values. The input file would be a .txt file with content similar to this:

A 12x12 .txt file used as input to the sudoku solver:

```
12
4
3
8.4C....A...
1....4..5CA.
.6....9..4.3
..C.A7.48...
3..7.B..CA..
6...1...3.82
B4.5...A...1
..59..B.1..7
...8C.61.2..
7.B..1....4.
.814..C....A
...B....78.C
```

When reading the contents of the input file, the students would likely store the values in an *int* or *char* array. The exercise text is indifferent to how sudoku board values are stored, giving students the choice of making an educated decision of what method they want to use. The chosen format (characters or integers) will matter when students are designing the solver, but when reading input, some students would chose to store them as integers and some as characters. The feedback in Game of Exams then has to test for both the different solutions, or add some constraints to the exercise. Doing so limits the autonomy of the exercise, and kills some of the potential for students ability to use their own creativity. If the exercise text added the constraint that the read input must be stored in an int array, it would be easier to create tests, as the format of the read solution is then known. But to try to keep the autonomy of the original exercise text intact, the automatic tests were made to test both a solution with integers, and a solution with characters. The tests then had to figure out what solution the students had chosen, then check if the solution was correct. This example is only one of many I experienced during implementation of exercises from INF1010.

The situation of where there are several possible approaches to solve one problem, often arises as exercises rarely require a specific output. Many exercises ask only that programs are made to to solve a problem, leaving the format of the solution up to the solver. If there are several ways of solving an exercise, the automatic tests also needs to test for several solutions. The author must choose between writing a lot of code for automatic testing, or constraining the exercise test, leaving students with less choice in choosing their own approach. As my goal was to implement exercises from the curriculum in INF1010, I would try to create tests for each weekly exercise set. But in the end few of the tests were completed with automatic feedback on all exercises.

To keep the exercises free of added constraints, writing tests were so time consuming that it mostly far exceeded the 5 hours I had available to prepare for seminar classes. Creating automatic tests is a complex and time consuming task. The tests need not only to check if the solution is correct or not, but should provide feedback when the users solution fails to solve the problem correctly. Giving good feedback means that the tests needs to predict what mistakes the student might make during the solution process. If the test catches the reason for an erroneous result, the feedback can be tailored to give relevant feedback. Finding different cases that lead to failed solutions is another time consuming task that makes the implementation of feedback very time consuming.

The only choice to make the task of creating automatic tests less time consuming is to add constraints to exercises. Or to choose exercises that are easier to test automatically. Some exercises does not suggest too many different solutions, and will then be easier to test.

The need for internally controlled runtime environment

The project was dependent on a web service at www.ideone.com to compile and run the programs submitted in Game of Exams. The possibility to actually compile and run programs, and get the result back in the browser became a core functionality of Game of Exams in the Beta. But the use of an external service for running the programs, has it's drawbacks. For one, Game of Exams functioned at the mercy the ideone service. Whenever ideone.com had downtime, a student could no longer run his/her program in the Game of Exams. The externally controlled service also had limitations to what libraries were available for each language. The method of testing written code in Java is quite common in all languages, and as a result there are ready made test frameworks for all widely used languages. In the development of Game of Exams, it was not possible to use external libraries, because the project is using an external service to run the Java programs. The ideone.com service only gives access to the standard libraries in Java. The option of building such a service for Game of Exams, was never considered, as I reckon this would have taken too much time. But for the project to be truly successful, especially in terms of making the creation of automatic testing as efficient as possible, the project needs a server that can accept program code, run it, and send back the result. If the server is controlled by the administrators of Game of Exams, external libraries can be made available on the server, which makes endless amounts of testing libraries available. Using a library like the JUnit Testing Framework would make available many ready made methods and test patterns that are being effectively used in large Java projects today. To use JUnit or a similar framework for the purpose of automatic testing of Java exercises, was never been tried in Game of Exams, but it's possible it would make automatic testing easier.

Feedback from students during seminar classes

The seminar classes were open to anyone, and was arranged in addition the other weekly seminar classes available. It was a class for students who wanted to experiment with the Game of Exams interface, solve weekly exercises, and get extra tutoring on any subject that were taught in the curriculum. Before every class I would try to implement the weekly exercises, so the students always had relevant content to practice during the class. But during the semester I learnt that the job of creating automatic tests made it impossible to implement all the weekly exercises. If the weekly exercises were not ready with automatic tests, they would be published at the site, with an available editor, but any mechanism related to feedback would not work. That includes the gamification mechanisms, like points, leaderboard and badges. Even though the class did not always have that weeks exercises ready with automatic feedback, students would frequently work with exercises that had been implemented with automatic feedback. One of the advantages of always publishing the weekly exercises were that I would get feedback from the students as they were solving them in an editor.

Having the exercises ready in the browser, ready to be solved with an editor and a way to run the program afterwards has certain benefits. When I have taught seminar classes as a teaching assistant in earlier years at IFI, I have not had a tool that integrated exercise, editor, and runtime environment into one tool. When students were doing programming exercises in my class, they might not bother to actually complete the exercise, they might just complete it half way on a piece of paper. As the whole process is simplified and gathered in one tool, it's easy and natural to make a complete solution. Other mechanisms that encourage students to complete the exercise, is the knowledge that the purpose is to have the solution tested, and to pass the tests the answer must be complete and correct.

As students were using their browser to practice programming problems, they gave vocal feedback on the exercises. I learnt that there is a very good reason that many of the exercises are not popular among the students. I got complaints that the text made no sense, it was impossible to understand the nature of the problem, the problem does not relate to a real problem etc. In some cases the students even found that an exercise had no correct answer. The quality of the exercises varied greatly, some were easy to understand, and had great educational potential. While others were hard to interpret the meaning of, and were teaching concepts that were wrong or irrelevant.

The reactions from students were especially negative when they were working on exercises that failed to relate to the real world in a realistic way. It is a common in the curriculum of INF1010 to use examples from the real world to teach programming concepts. The students then seem to expect the same real world relevance to be present in the exercises. A

weekly exercise set might revolve around a theme like "books", storing them, renting them out etc. Making the connection between a real library service and the exercises is short. But if the exercises suggests solutions that does not fit with the real worlds usual way of doing things, students quickly get confused, because they are accustomed to the taught connection between real world examples and programming. Students were often critical of exercises that asked for a specific action along the lines of "Create an array that can store 100 objects of the type *Book*". If this kind of exercise were to be given in the middle of an exercise set themed around building a library system, students will automatically try to draw the connection of what's the purpose of this exercise? If students can't see the purpose they will also struggle to understand what they are doing and why. In the seminar classes I had several students who sometimes just quit doing the exercise sets claiming that they "made no sense". This relates to the need for relatedness in SDT. When a students fails to feel related to the what he/she is doing, the student is likely to quit. When an exercise thwarts the feeling of relatedness, the students is likely to loose motivation to solve the exercise.

The occurrence of lousy exercises exists because making good exercises is difficult, they need to be refined over time. The usual way of creating and using weekly exercises at IFI, has no planned process of improving exercises. They are published once on PDF, students work on them as they wish, and the exercises are forgotten (until they are used next year). When the exercises are published in a web system like Game of Exams, the exercises can be changed and improved at any time when feedback are received from students.

Monitoring received answers

Game of Exams also presents a unique way to get feedback on the exercises through monitoring student answers. Answers are sent to www.ideone.com, and by logging on as administrator on the ideone account, all programs that are delivered is available. The sender of exercise is anonymous, but the content can be read. During the beta I found this to be valuable tool to learn how students respond to an exercise and what misinterpretations might occur when trying to solve it. Doing some programming mistakes while finding the right solution is natural, but sometimes misinterpretations of the exercise text can cause students to work on solutions that leads nowhere, and is basically a waste of their time. Observing the received answers from students after publishing a new exercise set, is a good way to catch unclear text formulation and other faults in the exercise text. Analysing student answers can also be done at any time, not only during the seminar class. Many students would work on the exercises in Game of Exams without participating in the seminar classes at all. In fact most of the users in Game of Exams were people who never showed up in the seminar class. Monitoring their answers a good way to receive feedback on the exercises from a large audience.

6.6 Latest version of Game of Exams

The development process has gone back and forth, and in this last section of the project chapter I will quickly summarise the state Game of Exams was in when development ended in April 2013. Game of Exams was a work in progress through the whole period, but even though there are always improvements to be made, I had to draw the line at some point. What is presented now is the state of the project after further development stopped in April 2013.

Learner interface, last version

The part the webapp where users actually solve exercise sets, has maintained much of the look shown in figure 5.6 on page 37 from the alpha version. Since then visual improvements has been made, as seen in figure 6.7 on the following page. Following the exercise text is the view users spend most of the time, where they answer the exercise. Figure 6.8 on page 71 shows an interface where users submit their program by pressing the "Test" or "Run" buttons, and the program output will show below the field called "Java7 result:".

The admin interface

The admin interface was probably the one that was redesigned the most times. As this is the interface where teachers create exercises, it was constantly changed as I was implementing different kinds of exercise sets in Game of Exams. In figure 6.9 you see that there is an editor available to write exercise text in markdown. There are also four tabs that are used to create code related to the exercise. "Code before" allows teachers to paste code at the beginning of a users solution. "Code after" allows teachers to paste code at the end of a users solution. This was made to facilitate user solutions to be only "partial" programs. The extra code will make sure the user solution is runnably, the most common usage for this is to create a main method and run a second class that incapsulates the users solution.

The tab called "Prefilled code" will be placed in the editor when users see an exercies for the first time. This is useful to start users off with some hints or a skeleton that can be used. It's used to ensure that users use variables that needs to be used for testing.

The code used for testing, is of course written in the "Tests" tab. This is what is often referred to as automatic evaluation, tests or feedback. This is also the tab that exercise creators are most likely to spend their time in.

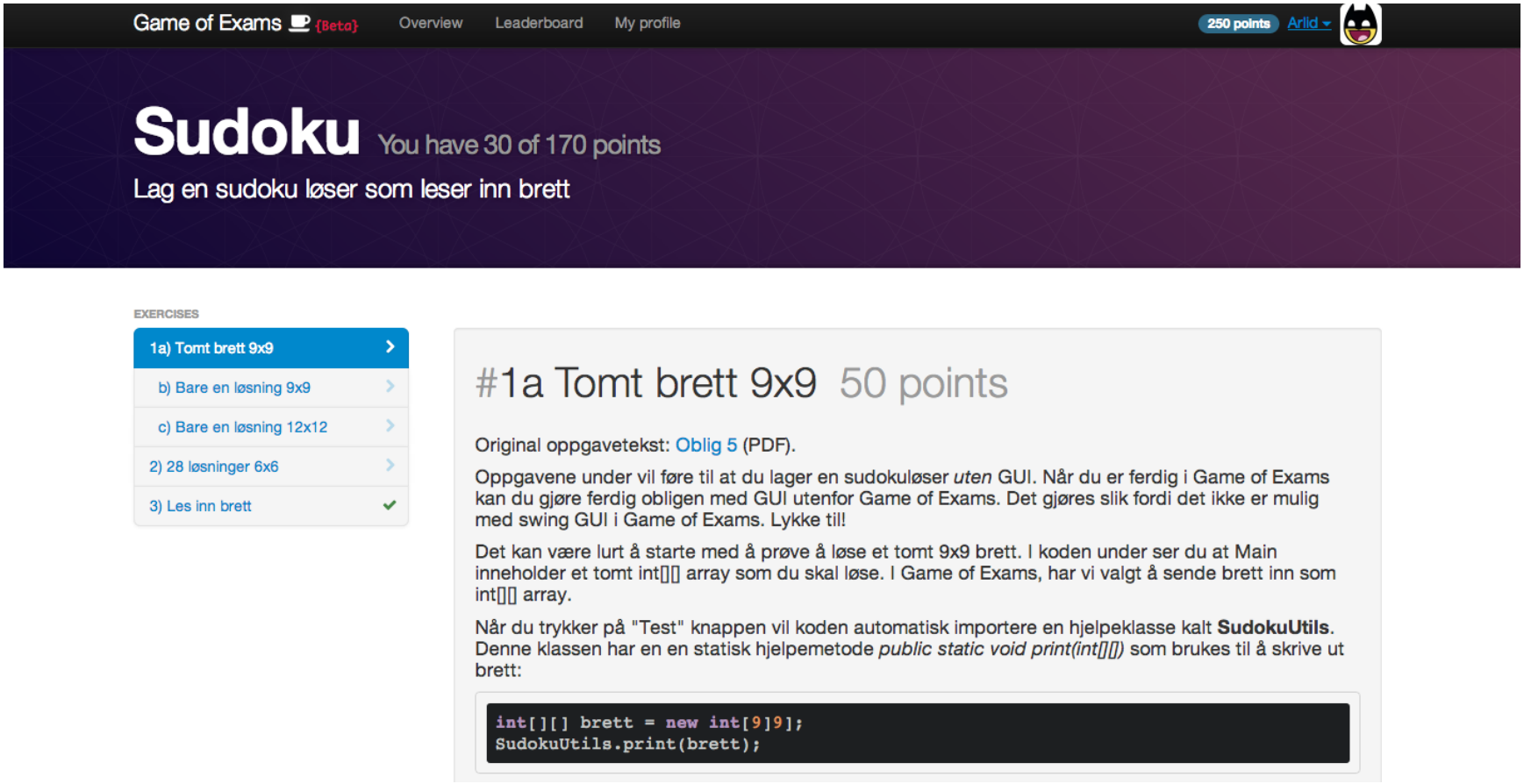

Figure 6.7: Game of Exams represents an exercise set entitled "Sudoku" to the user

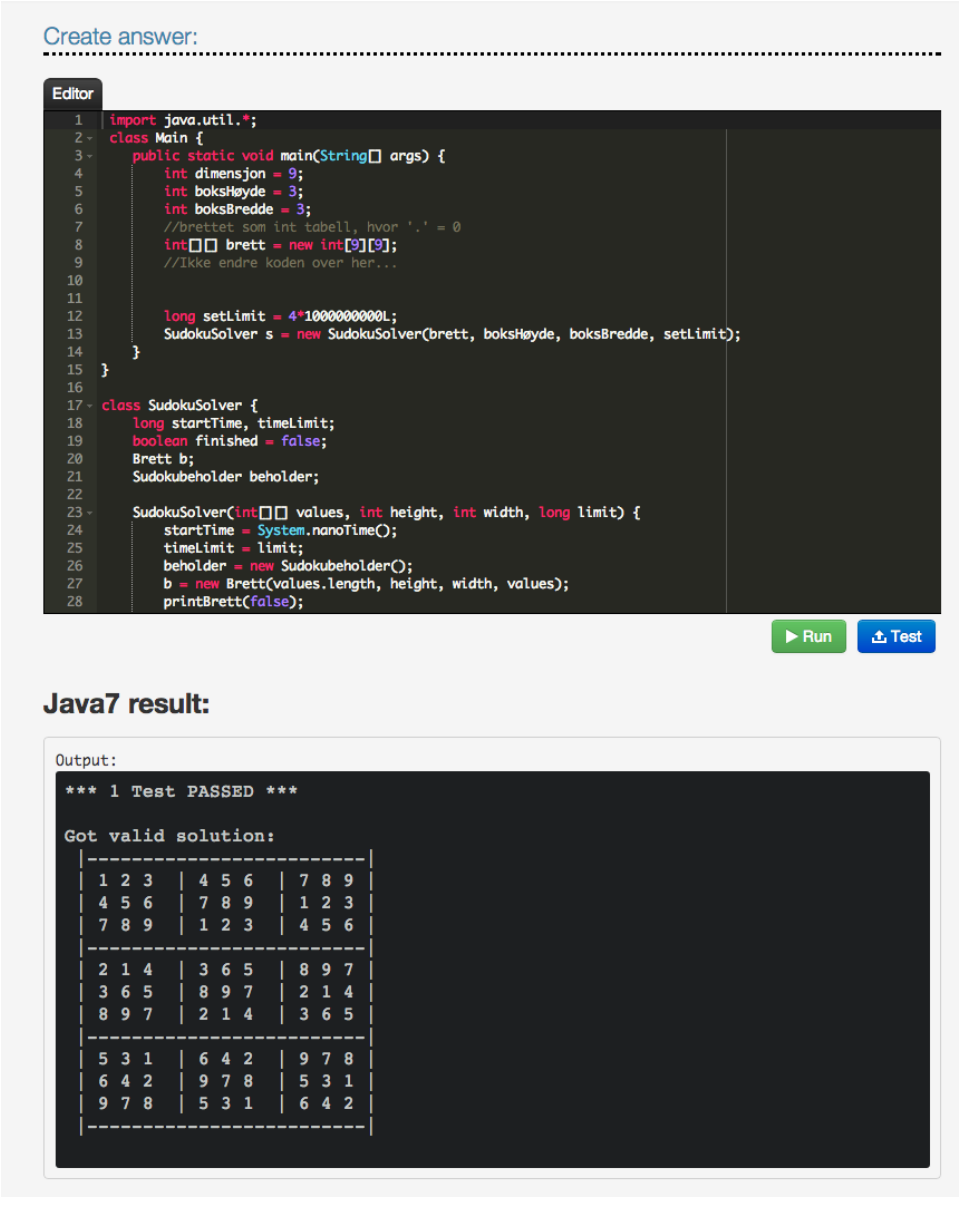

Figure 6.8: Final version of editor and output in the learner interface

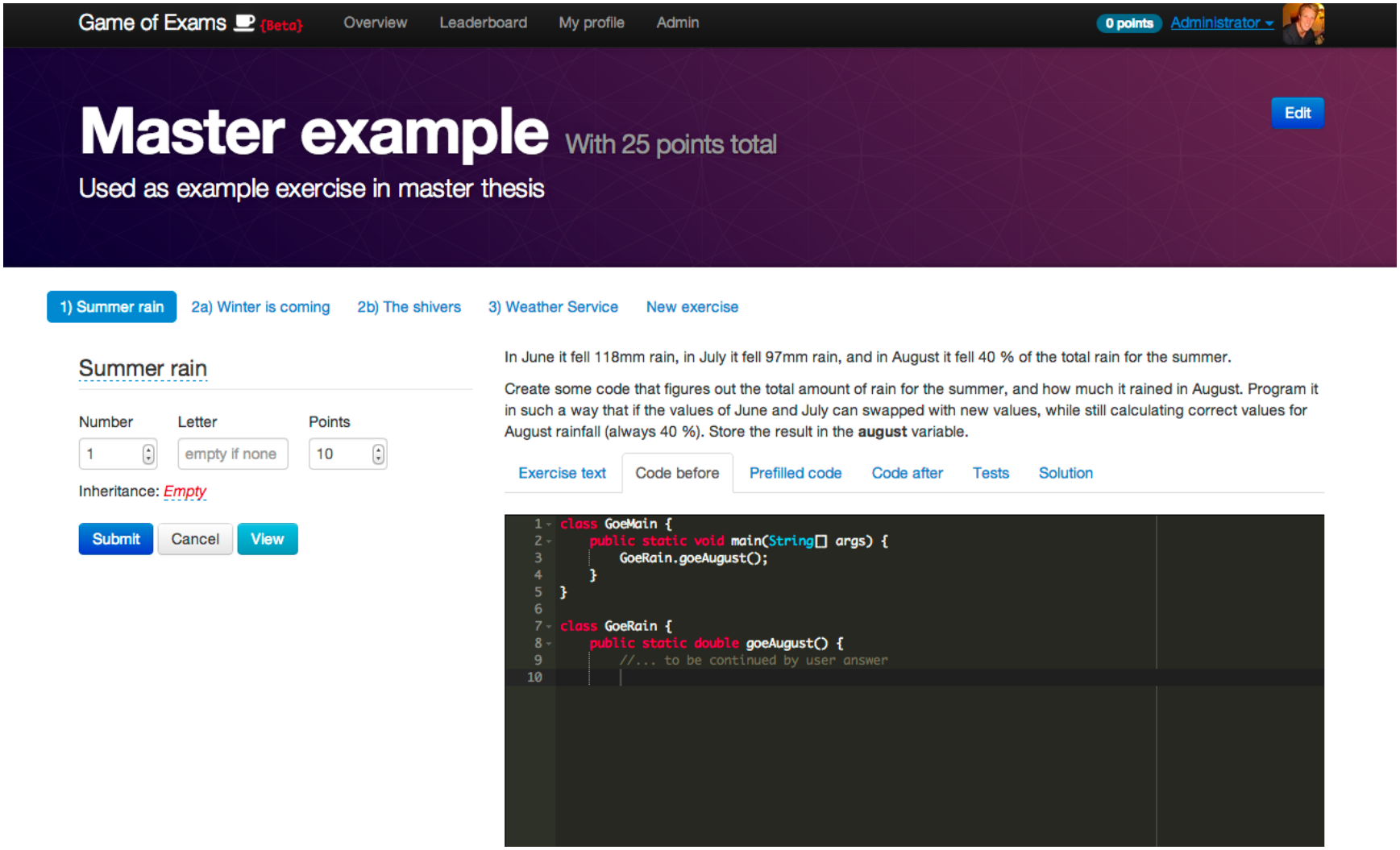

Figure 6.9: Last version of admin interface

Gamification elements

There were only minor changes to the gamification elements during Beta. Users personal avatars were added next to usernames on the leaderboard. This was done to make it easier for users to identify with other users on the site. Avatars makes it easier to recognise users as it adds a visual representation of a user. The final version of the leaderboard is shown in figure 6.10

The page that displays achievements remained unchanged since the alpha version and can be viewed in figure 41.

What is striking when I review the evolution of the webapp is how the reward system has remained almost unchanged since the start of the project. The whole challenge in the beta version has been to make the content seem meaningful to the users. That process has been highly dependent on automatic feedback and the challenges related to that.

As I was only one person to work as a developer I depended on other frameworks to take care of critical aspects in the application process. The fact that I used a service (ideone.com) to run programs for Game of Exams was critical to be able to do automatic program feedback. But this also ment that I had no control over the way the programs were evaluated. This limited my access to test libraries that are likely to make the process of automatic feedback easier.

Continued efforts would have to involve a way to have increased control over the environment programs are being executed in. This would allow for a more efficient way to create exercise feedback, which is the most difficult and time consuming aspect of Game of Exams.

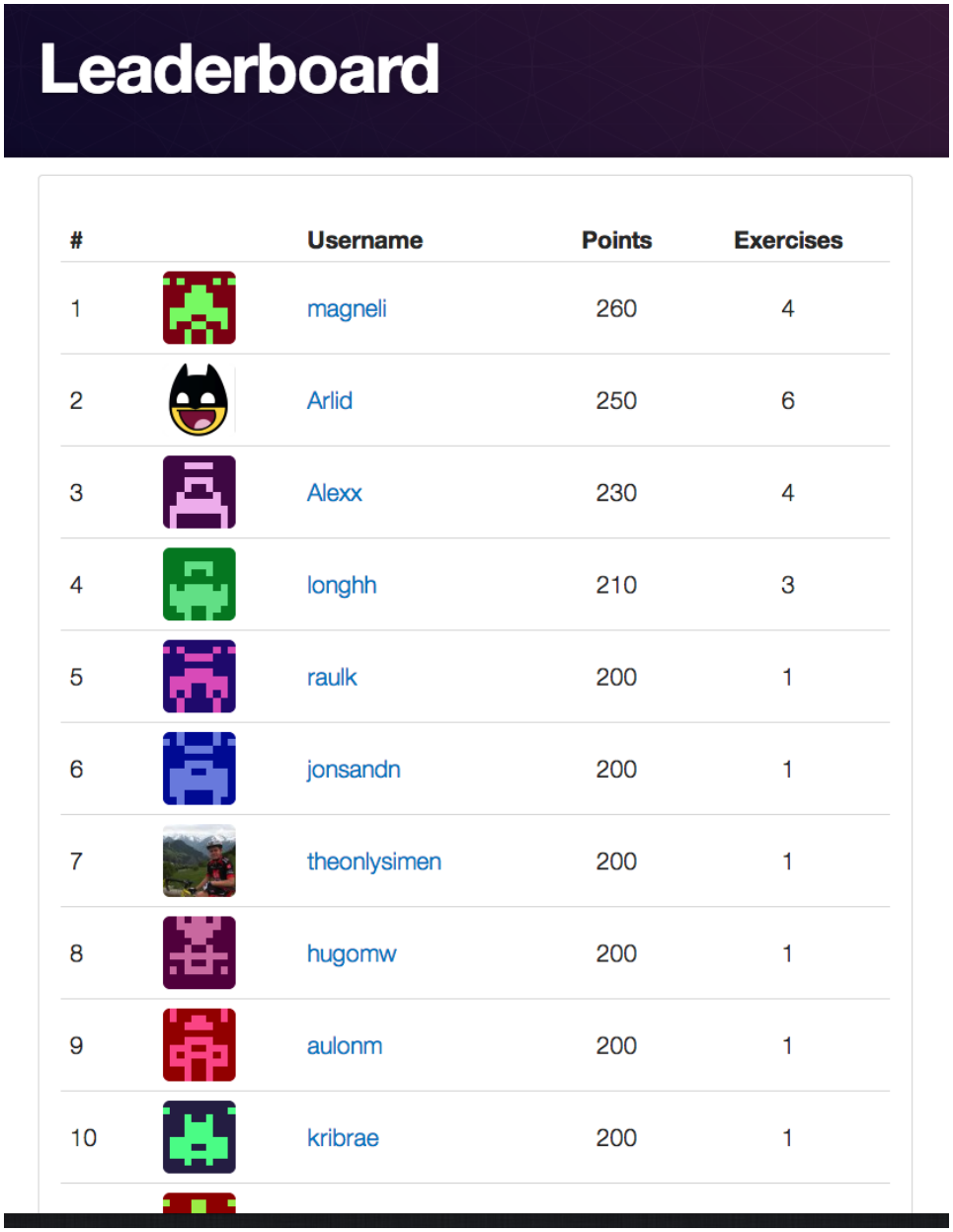

Figure 6.10: Screenshot from the leaderboard in July 2013

Chapter 7

Final results and analysis

After the alpha version, Game of Exams was implemented and used in INF1010 from January 2013 until the exam in July 2013. The seminar classes that were used as a concurrent evaluation tool, were ran from January to mid April. At the end of July, Game of Exams had 201 users, where the course INF1010 had around 430 students that qualified for the exam.

Tools from google analytics were used to gather data about the webpage during this period. The graph visible in 76 illustrates that engagement spiked heavily when new content was implemented. In march the webapp had most engaged users, when I released mandatory exercise 4 in Game of Exams.

Mandatory exercise 4 remains as the most engaging content released in Game of Exams. During that period over 3500 submitted answers was evaluated and given automatic feedback. The students participating chose to use Game of Exams without any pressure from the course administration. The test framework for mandatory exercise 4 was released as standalone code which students could download and use while solving the mandatory exercise. Yet many chose to do so in Game of Exams, using the gamified programming environment instead of their own programming environment.

The automatic tests for mandatory exercise four took about 40 hours to create, which says something about the amount of work required for creating successful feedback.

7.1 Summary of the design process

First a proof of concept implementation was made, the Alpha phase of the project. This period had a few goals that had to be examined before continuing the project. One of the first things that had to be assessed, was to see of a I as the only developer, would be able to find and use available technology to create this artifact within a few months. Developing the

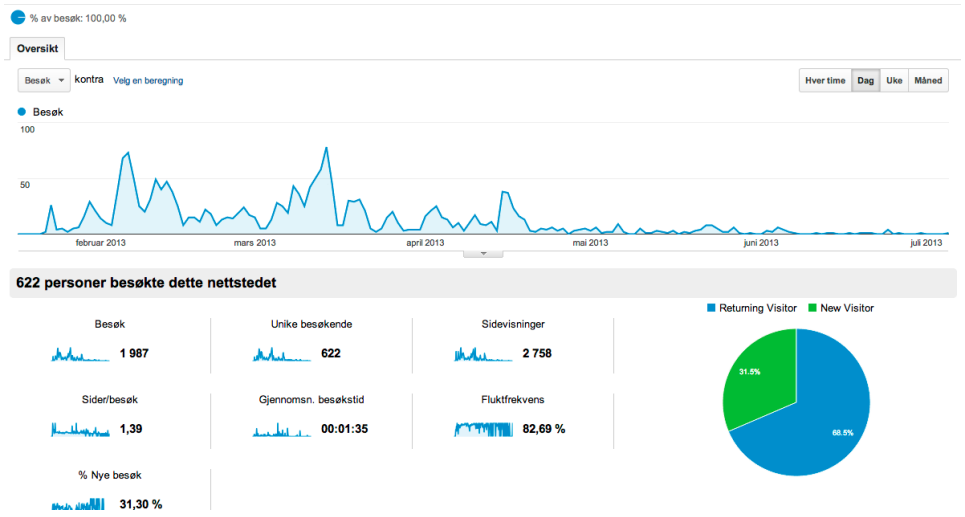

Figure 7.1: Data from Google Analytics

alpha version had promising results, the artifact was able to incorporate enough gamification and functionality to be tested on a course. The second test for the Proof of Concept was seeing the students reactions, getting their feedback, and experiencing the way the artifact worked as a teaching tool. The concept seemed to be interesting to students, who voluntarily used it both in the seminar classes and on their spare time. At the end of the alpha, Game of Exams had been used by around 2/3 of the students, even though only 15 % of the student mass showed up in class. This means that most users found the webpage themselves following a link on the course web page. During the period the alpha had 58 unique users, 129 unique answers, and a total of only 13 exercises.

During this period the students activity on the site was watched closely, to see how they interacted with the site. Around half of the users solved some programming exercises on the site, with the other half of users probably just checking out the page out of curiosity. Engagement seemed high for the 5 users racing to the top of the leaderboard, who seemed to work through the exercises at a determined pace. While observing students during seminar classes they expressed enthusiasm for the gamification mechanics, especially the part of obtaining badges. After the class they would continue to solve exercises until everything was completed. I think there is reason to believe that the webpage did cause extra engagement in the users, as all users had the exact same exercises available in pdf format. In the Alpha version the answer editor was a horrible way to create programming code in, as it was only a standard html5 textfield. If the gamification mechanics had no value, I would expect few students to bother using the webpage at all, because answering in Game of Exams required extra work (using the textfield combined with markdown). Even so the students chose to deliver their answers at the webpage, hopefully because of the game mechanics of the webpage was in some way meaningful to the students.

When designing a gamification platform for education it was somewhat helpful to get some suggestions from earlier research on the subject Muntean (2011), Erenli (2012). This research is limited in application, as they are only theoretical analysis of possible scenarios. And because there exists some very successful examples of gamified websites that teach already programming, it's more helpful to examine the game elements that work well for those examples. But the most critical to a successful gamified application, is to ensure that the gamification is meaningful in it's social context. Sebastian Deterding, the leading theorist in the field of gamification, sketches out a theoretical model based on SDT, but concludes that the model leaves much to be asked for (Deterding, 2011). Nicholson takes it a bit further and creates a theoretical framework to be utilised for meaningful gamification (Nicholson, 2012). While Nicholson mixes some other theories in the framework, he also relies heavily on SDT theory, especially Orgasmic Integration Theory (OIT) which they suggest is useful to balance the use of external rewards in the gamified application. Following these theorists, SDT is used to ensure that the educational activities in Game of Exams are as meaningful and fun as possible for the university students.

As the project went on, the two theoretical models were used on different levels of the application. FBM was used much in the start to measure what kind of functionality is needed for the students to reach the activity threshold to do exercises. Because the part of motivation is covered so deeply in other theories included in this thesis, the most useful part of the FBM, was the elements of ability, which he often referred to as concerning simplicity. The elements from FBM concerning ability became important to find weaknesses in the design. It would not matter how motivating the gamification was, if the design did not give users the ability to use the website. With this in mind, I specifically focused on simplifying the activities most relevant to the main user groups of the application: Students and Teachers. In the next sections I will go specifically into how ability was facilitated for these two groups in Game of Exams.

7.1.1 Students ability to solve programming exercises

What we want students to do is solve programming exercises. After reviewing the FBM's elements of ability, I found three of relevance to Game of Exams. That being – time, brain cycles, and non-routine. Money were no threat to ability as Game of Exams is free, and physical effort could also ruled out as students are only sitting by their computers. The element of social deviance was of little threat to students ability, as I have no experience of students having to do any activities in Game of Exams that is in some way is not socially acceptable.

The element of time were considered, and during development I tried to make sure that the webpage has very few processes before requires them to part with any more time than they have to. The process that

takes most time is probably creating a user account, but even that process has been designed to involve very few steps. As this is the beta, many settings are standardised for most users, so they don't have to spend time giving information about courses they are taking, programming language preferred or anything like that. Even the process of getting a personal avatar is automated, using a service [1] that loads an existing avatar, or generates a brand new one for the user. The biggest threat to time use is probably the loading time of the webpage which at current time is quite a bit longer than one would expect from a professional service. But this is not a big issue, because it's a result of the limited resources used for this project (only one developer, and a free server from www.meteor.com). A project with more resources would likely have no problem designing a fast responsive webpage.

The element of brain cycles might seem a bit weird in the context of programming, it's pretty much a requisite for doing the activity. Still, I would interpret this element of FBM's simplicity model as trying to ensure that the main activity of the site is not confusing in any way. The interaction design needs to be solid, and users should not be left guessing what icons and other design items on the site means. To facilitate this Twitter Bootstrap was used, which is a design framework that has many tools help designers create easily recognisable buttons, menus, contextual messages, dropdown boxes etc. If users quickly recognise the design of the webpage, they will be able to easily focus on using their brain cycles on the important task, which is solving exercises. The element of brain cycles role in simplicity, was in the thesis used to ask myself questions if users were able to focus on solving the exercises, instead of thinking about other elements.

The element of non-routine was addressed as as Game of Exams tried to do create a programming environment that was close to the programming environment students were used to. By adding a code editor with syntax highlighting and other functionality coders are used to, students hopefully got to use something similar to what their everyday programming environment looks like. In the alpha version the webpage had users answer programming exercises using a normal html5 textfield, which is far from their routine of using a programming editor.

In the alpha version the element of social deviance played a role, as students had the possibility of "cheating" as there was no automatic feedback when submitting answers. The answer feed exposed all answers publicly, hoping that users who posted empty answers would see that their behaviour was not the norm as most users did not cheat. The answer feed was later disabled as automatic feedback was integrated in the Beta version.

[1] A commonly used avatar service seen on many sites in the programming community: www.gravatar.com

7.1.2 Teachers ability to create programming exercises

A very important aspect of the artifact is the admin interface. This is where teachers create the exercises that are published to the students. Getting this part right is crucial to the success of the platform, as we want teachers to have the best possible tool to create fun engaging exercises for the students.

Creating a good admin interface proved to be much more challenging than creating the student interface. This was because of a few reasons: 1) There are several good real world examples that exemplify working gamified programming environments. But these applications usually design their courses from scratch, probably not using any "admin interface". The only site that exposed some kind of admin interface was Code School, which did provide some hints in the starting phase of the admin design. 2)

I quickly realised that the admin interface was the one part of the application that needed the greatest amount of flexibility to fit the unique context of the artifact. The admin interface was changing much more often, as the admin interface had to fit with the demands of implementing curriculum material into the platform. The fact that the context was Java programming, proved to create extra difficulties during development of the admin interface.

7.2 Self-determination theory

The main challenges during development of Game of Exams has been creating an artifact that fits the context of a university level programming course. With the technical tools available today, together with gamification design methods, it's quite easy to create something that looks like an impressive learning platform. But it is the psychological sides of the artifact that are interesting in the end. Is the gamification meaningful? Does it fit the context it is designed for? Does it create engagement for learning programming curriculum? These factors are hard to try without actually implementing the artifact in a real social context. During and after development, the concepts of self-determination theory, are invaluable in trying to answer the questions above. I will in this section outline the role of self-determination theory for the project. This can be seen as an application of the theorised concepts from the articles of Deterding (2011) and Nicholson (2012). In their articles they suggest that SDT theory is a useful framework in creating what Nicholson calls meaningful gamification, and Deterding calls situated motivational affordances of game elements.

Facilitating the needs for autonomy and competence proved to be highly difficult tasks in developing the artifact. When I started the project expected to use Ping Zhang's design principles to facilitate these needs. But as I developed and used the artifact I experienced that facilitation of psychological needs are highly context dependent. In Zhang's design

principles for ICT use, her primary concern in supporting autonomy is expressions of self-identity. This is facilitated by allowing users to express themselves through choosing avatars and customising the application to their liking. Although this design principle is important in many cases, also in Game of Exams, I find it to be of little relevance compared to other more important threats to autonomy that are unique to the context of programming, gamification, and education. This supports Sebastion Deterdings theorised claim that motivational affordances is situated. I find this to be true not only for gamification, but also highly situated in terms of the non-game context that is gamified. The context for this project is creation, implementation, and solving of university level progamming exercises. For this specific context, there are specific subtheories of SDT that are relevant, where especially education has been researched heavily by SDT theory.

With the principles of SDT theory in mind, these were specific problems that arised during development and testing of Game of Exams. The pure technical problems aspects are usually solved, but the hard thing is to design solutions that work well in our context. In the context of teaching, creating, solving and gamifying university programming exercises, these were the problems encountered:

7.3 The price of autonomy and competence

Cognitive Evaluation Theory (CET) is a subtheory of SDT that is specifically concerned with intrinsic motivation. As mentioned in the theory chapter, intrinsic self-determined behaviour is critical in the context of education. Autonomy and competence is shown to be critical in fostering intrinsic motivation. Unfortunately it is very easy to create an environment for students where these needs are not suppported. In Game of Exams, the gamification is dependent on automatic feedback that serves two purposes. One is to determine if the exercise is correct or not, which leads to activation of game elements. After that is done it's easy to forget the second role of the feedback, which gives a message to the student of his or her competence. Studies has shown that positive performance feedback enhance intrinsic motivation, while negative performance feedback diminishes it (Deci, 1975). In my review of other gamified programming platforms, positive feedback was a key component in all the websites reviewed.

But the task of providing timely, relevant and positive feedback is easier said than done. Often when implementing exercises, I would face the problem of having to write many different tests, to check the students answer for a range of possible solutions. Many programming exercises from the INF1010 curriculum had more than one ways to solve the same problem. This became a great problem when writing automatic tests, a large amount of time were spent creating tests that would recognise different solutions. And when all solutions are tested, the job of creating

feedback to failing solutions remains. When the user submits an incorrect answer, the exercise creator needs to create feedback that is informative of why the solution is failing, while giving a helpful and positive message of what's missing. This requires both programming effort, in the sense that the automatic tests has to be precise to figure out *what* the student is doing wrong. This was built into the test framework in the way of a detailed error message.

Monitoring user submitted answers was a good way to see if the tests were giving back helpful feedback. The case was often that it's hard to foresee what kind of misinterpretations occur, or what kind of bugs they might create submitting faulty solutions. The good thing is that by reviewing answers submitted, I could improve the automatic tests if I found that there were common mistakes frequenting that did not receive good feedback. To explain I will give an example. Let's say there is an exercise to create a data structure to store objects. This data structure should have method to add an object to the list, and when the object is added the size of the list should increase. During automatic testing I would create a test to check if the list actually increases in size after an object is added. If this does not happen the solution is faulty, and the tests should give back some informative feedback. A discouraging and maybe confusing message would be to return this feedback: "Error: wrong list size". This gives little information about why it has happened, and does not try to state the message in a positive way. I could rather give feedback in this manner: "Ooops, the list size is wrong after adding an object. Are you sure that size increases as an object is added?".

To allow informative feedback the testing framework was designed to always include a detailed message of why a test failed. The size test can fail under different conditions, maybe the list size is wrong after *removing* an object? The feedback should then give back a message describing the reason the test failed, giving the student an opportunity to resolve the problem. And if the feedback is delivered in a positive undertone, it's likely that the student feels competent and keeps solving the exercise with the bonuses intrinsic motivation. The problem with these detailed feedback messages is that it takes time to code them. As feedback details the origin of the size error, the exercise author must write more tests to determine the nature of the programming error. But if exercise authors don't take time to create precise, positive, information feedback they risk wasting their time entirely. Without precise automatic feedback, students risk getting feedback that does not fit their situation, which can seem very confusing, and maybe even unfair. The worst case would be if the automatic tests does not recognise a valid solution to the exercise problem. Given incorrect feedback, it's likely that the students feelings of both competence and autonomy is thwarted. If the feedback feels unrelated to the users answers, the user might feel like the results of his actions are out of his control. To feel autonomy users must see that when they change their code, the automatic tests respond with feedback that relates to what is happening in the students program.

According to CET, feelings of competence must be accompanied by feelings of autonomy to be effective in increasing intrinsic motivation. The student needs to feel an internal perceived locus of causality, for competence to be experienced. To facilitate this it helps to phrase the feedback as hints or suggestions. If the feedback tells the student directly how to solve the error, he/she is likely to feel little competence when solving it, because he/she does not have the feeling of being the originator the behaviour that lead to the solution. It's about providing opportunities for experiencing competence in an environment that does not feel controlling.

The most flexible automatic tests created in Game of Exams was made for the fourth mandatory exercise in the course. A screenshot from the feedback can be seen in 6.6 on page 61. The wording of the detailed messages is not as suggestive and positive as I have advocated above, but there are other facets here that support both autonomy and competence. The feedback demonstrates competence and achievement by displaying a list of passed tests rewarding points. The feedback displays several informative messages of where the program fails. The student is the agent in choosing what test to work on solving, and in using own capabilities to find the bug. The feedback gives just enough information for the student to look for the bug.

The feedback designed for mandatory exercise 4 gives students opportunities feeling to competence. By starting with a "clean sheet" of no passed tests, opportunities for displaying competence is displayed clearly in the visualisation of their current progression. As the students solves test after test, their competence and achievement is displayed by the feedback. Passing a test also led to other effects displaying competence and achievement. Points are awarded, and the student advances on the leaderboard. The leaderboard is an opportunity to saturating the need for relatedness, where students behaviour is set in relation with valued others. Who are valued others is determined by the social context. Since the students are taking the course together with other students they know in the real world, they hopefully recognise other users in the context of Game of Exams. To increase the chance that users recognise people from their social context, avatars were added to the leaderboard. The combination of an avatar displayed next to the username, hopefully increases the chance that the user will see a relation to other users on the site. Even if the users does not know each other, it's easier for them to relate to users in the context if they can relate a username to an avatar picture.

As the student relentlessly works through all tests in mandatory exercise four, suddenly all tests pass, and the student has solved the exercise. This is the pinnacle of achievement, and Game of Exams should respond in a way that gives pleasure to the user. Next to getting the points for passing an automatic test, the now receives a rare reward, a badge achievement. The automatic tests prints out a surprise to the user, an ASCII art picture of Batman 7.2 on page 84. Seeing the picture of Batman shine

from the output, is hopefully a joyful sight to the hours of struggling with tests and problem solving. Spicing exercises solving up with these kind of surprises is related the pleasure/pain element of motivation in FBM. The immediate of response of a cool Batman picture in this setting hopefully results in a pleasurable response in the user. By spicing the automatic feedback with these kind of pleasurable surprises, the experience of solving exercises is less dull. Students are reminded that there is in fact a human that created the automatic tests.

The relieving feeling of completing the exercise was displayed one evening I was working with the lecturer in INF1010. As he solved one test after the other he struggled to pass the final tests because of a flaw in his solution. After hours of hard work he victoriously raised his hands in the air, smiling. He had done it, and the face of ASCII Batman affirmed his efforts. In the context of university education this kind of things might seem silly, but I think experiences like this makes education more fun.

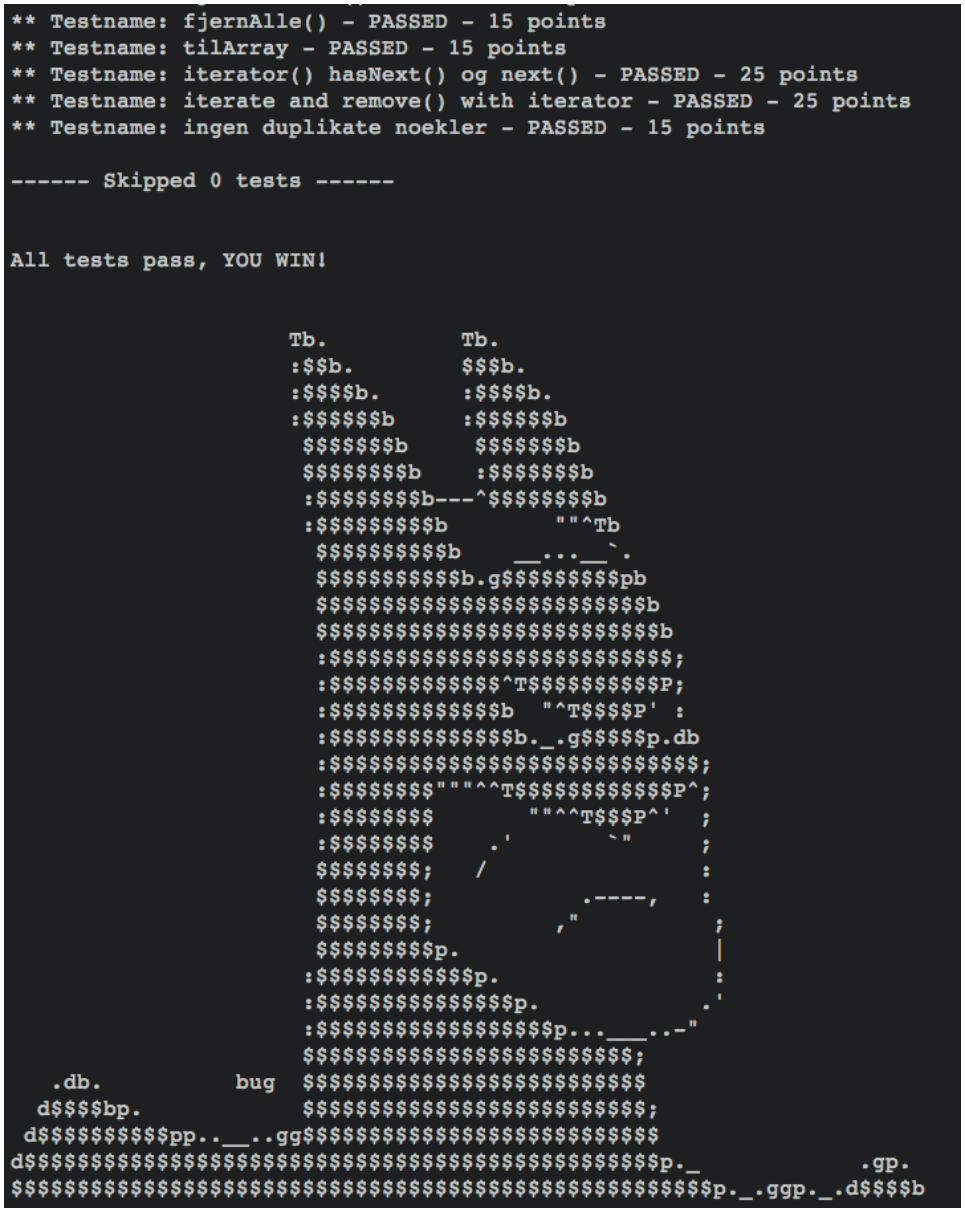

Figure 7.2: Batman ASCII art displayed when the exercise is completed

Chapter 8

Discussion and conclusion

8.1 Theoretical implications

Creating Game of Exams has been an interesting way to explore gamification and programming. Even though the concepts of SDT are applied in a specific context, the results from this thesis shows support to the suggestions made in earlier articles (Deterding, 2011, Nicholson, 2012) that SDT can play an important role in gamification efforts. The application of the SDT framework to analyse the gamified artifact, gives a unique possibility to understand non-technical sides of the artifact. As gamified applications are so fragile, in the sense that it takes very few weaknesses before aspects of the application does not support users needs, and the artifact fails to create engagement for it's users.

SDT is a great framework for understand the artifact in terms the underlying features of the application and how they affect the users. SDT has perspectives that allow creators of technology to understand users needs in a very different way than designers are used to, and in Game of Exams it seems to work very well. Without any planned strategy of how to use any of the theoretical frameworks, I have come to see that the theoretical frameworks fits to be used in different stages of a project. During development of core functionality, important questions are raised because of SDT theory. When the functionality is developed and put to use, FBM has been applied as kind of as frosting on the cake. To make the end product better. Where SDT theory is used to analyse the core functionality, FBM principles are used to make sure they are the most effective when designed. FBM is useful to see where you want to focus your efforts on persuading users, and trigger more activity.

In the context of using gamification in the artifact, the different ways the theoretical frameworks has been put to makes a clear distinction to how they should be applied in a design process. SDT guided important decisions on what game elements to use, based on their theorised effects according to SDT theory. The leaderboard which is not seen in the other gamified programming services, is included in Game of Exams because

of the need for relatedness and competence. Because the social context is different in Game of Exams, than these worldwide services, the chance that users experiences relatedness by interacting with valued others is higher. This is one specific way SDT has influenced Game of Exams, by guiding early decisions relating to core functionality. Where SDT helps to develop core functionality, FBM's most used role has been to improve effects of already implemented features of the webpage. This could be improving simplicity of the design, or trying to affect users emotions better by creating anticipation (hope/fear aspect of FBM). FBM was used to improve the design of the features that was implemented in guidance with SDT research. Both frameworks are used to influence users behaviour, but on a different scale. Where SDT helps the designer focus on what users feel on deeper levels, FBM helps guide superficial design decisions.

The application of self-determination theory sets the bar high for the gamification mechanisms. But as the course content is used in Game of Exams, it becomes part of the artifact. While doing that I would constantly subject course content (programming exercises) to the same scrutiny of the mechanisms that try to secure meaningful gamification. Doing this underlines the importance of making meaningful programming exercises. We should try to provide students with challenging, autonomous exercises that students can relate to. That would be meaningful education.

Creating meaningful content is a difficult job, and it's even harder to do it on a platform that requires automatic feedback. But the work processes involved in transferring programming exercises to Game of Exams work well to expose weaknesses in the exercises. Through the methods of creating informative, positive feedback, exercise creators has to examine the exercise again and again, and the exercise is measured against the precise demands of automatic feedback.

8.2 Future work

The theoretical framework established for Game of Exams allows us to explore gamification of education in a way that keeps a solid focus on the motivational effects of the artifact. There are still many ways this could have been explored, that I did not have time to do in my reseach. The thesis ended up exploring mostly the facets of motivation related to autonomy and competence.

But this kind of artifact has an amazing potential to futher facilitate relatedness. If someone were to continue the project it would be very interesting to see how developers could make the webpage more socially relevant to users. In 2011, master student Guro Johansen wrote a thesis about the need for a social platform for informatics students at IFI (Johansen, 2011). It would be interesting to see her research integrated into Game of Exams, which would make it a platform for learning and socialising.

I would encourage further use of action design research as the methodology for a similar project to Game of Exams. It has been very natural to design the artifact in relation to the organisational context. Without the concurrent evaluations done throughout the project, I think it would be hard to identify the real problems experienced during development. I also think ADR allows projects that has few resources, to achieve more on a shorter time period. As I was able to evaluate the artifact during development, made it easier to keep the development efforts relevant to the most important problems.

My work is highly influenced by the fact that INF1010 teaches Java programming. I believe that the difficulty of creating automatic tests varies greatly from what language to be used. Java, being a class based, object oriented language makes it hard to create automatic feedback for exercises with high autonomy support. Using a scripting language like Ruby, Python or Javascript might make the process of creating a test framework much easier. This might explain why these are the languages that are frequently seen in services that teach programming in an interactive way. Java on the other hand, is not taught in any of the professional services using gamification to teach programming.

8.3 Conclusion

Gamification is a tool that is easy to use but hard to master. In the start of the project gamification tools were designed and added to Game of Exams. By following examples of gamification seen in similar applications, it's easy to create a reward system for your application. But the real challenge lies in making the implemented gamification mechanisms meaningful. This relies on the context that is gamified.

When programming exercises is the content put into such a platform, the success of the gamification relies on the way the exercises relate to the users. In this thesis the biggest challenge to facilitate this has revolved around the artifacts ability to give meaningful feedback.

The automatic feedback system is the functionality that has the most power to decide what kind of content can be used, and if users is engaged in using the content. The challenge for future work must be to create effective feedback systems related to the content that is put into the gamified artifact. Programming exercises will need to have language specific feedback systems, which can be used by exercise creators to create fun autonomous exercises that can be automatically evaluated.

In the discussion of how to achieve meaningful gamification,Deterding (2011) and Nicholson (2012) discussed the importance of context dependence. Nicholson criticises organisationally context dependent gamification and suggests a user-centered approach. In this thesis I have found that the possibilities of creating user-centered gamification also depends on the organisational context. To be able to create exercises that support

autonomous opportunities for exercise solving (user-centered), the organisation must create tools that allow creation of good exercises. This can be facilitated by an organisation that provides good resources for automatic testing and creative exercise creation.

This thesis has shown some initial success to the attempt of creating a gamified teaching platform for university curriculum in programming. But the artifact itself needs to be refined with better tools for automatic testing. If that is achieved I believe it's possible to create a website that will create a new level of engagement in the education informatics students.

Bibliography

10 new gurus you should know - BJ Fogg (1) - FORTUNE, 12. URL http://money.cnn.com/galleries/2008/fortune/0811/gallery.10_new_gurus.fortune/.

C. M. Bachman and C. Stewart. Self-Determination Theory and Web-Enhanced Course Template Development. *Teaching of Psychology*, 38(3): 180–188, June 2011. ISSN 0098-6283. doi: 10.1177/0098628311411798. URL http://top.sagepub.com/lookup/doi/10.1177/0098628311411798.

E L Deci and R M Ryan. The support of autonomy and the control of behavior. *Journal of personality and social psychology*, 53(6):1024–37, December 1987. ISSN 0022-3514. URL http://www.ncbi.nlm.nih.gov/pubmed/3320334.

E. L. Deci, R. Koestner, and R. M. Ryan. Extrinsic Rewards and Intrinsic Motivation in Education: Reconsidered Once Again. *Review of Educational Research*, 71(1):1–27, January 2001. ISSN 0034-6543. doi: 10.3102/00346543071001001. URL http://rer.sagepub.com/cgi/doi/10.3102/00346543071001001.

Edward Deci, Robert Vallerand, Luc Pelletier, and Richard Ryan. Motivation and Education: The Self-Determination Perspective. *Educational Psychologist*, 26(3):325–346, June 1991. ISSN 0046-1520. doi: 10.1207/s15326985ep2603\&4_6. URL http://www.informaworld.com/openurl?genre=article&doi=10.1207/s15326985ep2603&4_6&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3.

Edward L. Deci. Intrinsic motivation. *New York: Plenum*, 1975.

Edward L. Deci, Richard M. Ryan, and Geoffrey C. Williams. Need satisfaction and the self-regulation of learning. *Learning and Individual Differences*, 8(3):165–183, January 1996. ISSN 10416080. doi: 10.1016/S1041-6080(96)90013-8. URL http://linkinghub.elsevier.com/retrieve/pii/S1041608096900138.

Sebastian Deterding. Situated motivational affordances of game elements: A conceptual model. *...Using Game Design Elements in Non-Game ...*, pages 3–6, 2011. URL http://gamification-research.org/wp-content/uploads/2011/04/09-Deterding.pdf.

Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart E Nacke. From Game Design Elements to Gamefulness: Defining "Gamification". In *MindTrek'11*, Tampere, Finland, 2011a. ACM. ISBN 9781450308168. URL http://dl.dropbox.com/u/220532/MindTrek_Gamification_PrinterReady_110806_SDE_accepted_LEN_changes_1.pdf.

Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification. using game-design elements in non-gaming contexts. *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA '11*, page 2425, 2011b. doi: 10.1145/1979742.1979575. URL http://portal.acm.org/citation.cfm?doid=1979742.1979575.

D DiNucci. Fragmented Future. *Print*, 1999. URL http://elibrary.ru/item.asp?id=3696048.

Adrián Domínguez, Joseba Saenz-de Navarrete, Luis De-Marcos, Luis Fernández-Sanz, Carmen Pagés, and José-Javier Martínez-Herráiz. Gamifying learning experiences: Practical implications and outcomes. *Computers & Education*, 63:380–392, April 2013. ISSN 03601315. doi: 10.1016/j.compedu.2012.12.020. URL http://linkinghub.elsevier.com/retrieve/pii/S0360131513000031.

Kai Erenli. The impact of gamification: A recommendation of scenarios for education. In *2012 15th International Conference on Interactive Collaborative Learning (ICL)*, pages 1–8. Ieee, September 2012. ISBN 978-1-4673-2427-4. doi: 10.1109/ICL.2012.6402106. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6402106.

Kai Erenli. The Impact of Gamification-Recommending Education Scenarios. *International Journal of Emerging Technologies in ...*, 8(1):15–21, 2013. URL http://www.editlib.org/p/45224/.

Bj Fogg. A behavior model for persuasive design. *Proceedings of the 4th International Conference on Persuasive Technology - Persuasive '09*, page 1, 2009. doi: 10.1145/1541948.1541999. URL http://portal.acm.org/citation.cfm?doid=1541948.1541999.

J Hamari and Jonna Koivisto. Social motivations to use gamification: an empirical study of gamifying exercise. *Proceedings of the 21st European Conference on ...*, 2013. URL http://www.hiit.fi/u/hamari/2013-social_motivations_to_use_gamification-preprint.pdf.

Alan R Hevner, ST March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004. URL http://dl.acm.org/citation.cfm?id=2017217.

E Tory Higgins. Value from hedonic experience and engagement. *Psychological review*, 113(3):439–60, July 2006. ISSN 0033-295X. doi: 10.1037/0033-295X.113.3.439. URL http://www.ncbi.nlm.nih.gov/pubmed/16802877.

Guro Johansen. *Sosiale mediers bidrag til det psykososiale studentmiljøet ved Institutt for Informatikk*. Master thesis, University of Oslo, 2011.

Kuutti Julius and J Salo. Designing Gamification. *Marketing*, (May), 2013. URL http://herkules.oulu.fi/thesis/nbnfioulu-201306061526.pdf.

Daniel Kahneman, Ed Diener, and Norbert Schwarz. *Well-being: The foundations of hedonic psychology*. G - Reference, Information and Interdisciplinary Subjects Series. Russell Sage Foundation, 1999. ISBN 0871544245. URL http://books.google.de/books?id=3toRUh4L12EC.

Eric Klopfer, Scot Osterweil, and Katie Salen. Moving learning games forward. 2009. URL http://telearn.archives-ouvertes.fr/hal-00593085/.

Anelly Kremenska. Technology enhanced language learning: student motivation in computer assisted language learning. *...of the 2007 international conference on Computer ...*, pages 1–6, 2007. URL http://dl.acm.org/citation.cfm?id=1330690.

Dan Lockton. Persuasive technology and digital design for behaviour change. *Available at SSRN*, pages 1–17, 2012. URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2125957.

CI Muntean. Raising engagement in e-learning through gamification. *Proc. 6th International Conference on Virtual Learning ...*, (1), 2011. URL http://www.icvl.eu/2011/disc/icvl/documente/pdf/met/ICVL_ModelsAndMethodologies_paper42.pdf.

Scott Nicholson. A User-Centered Theoretical Framework for Meaningful Gamification. In Crystle Martin, Amanda Ochsner, and Kurt Squire, editors, *Proc. GLS 8.0*, pages 223–230, Pittsburgh, PA, 2012. ETC Press. URL http://scottnicholson.com/pubs/meaningfulframework.pdf.

Andrew K Przybylski, Netta Weinstein, Richard M Ryan, and C Scott Rigby. Having to versus wanting to play: background and consequences of harmonious versus obsessive engagement in video games. *Cyberpsychology & behavior : the impact of the Internet, multimedia and virtual reality on behavior and society*, 12(5):485–92, October 2009. ISSN 1557-8364. doi: 10.1089/cpb.2009.0083. URL http://www.ncbi.nlm.nih.gov/pubmed/19772442.

R M Ryan and E L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *The American psychologist*, 55(1):68–78, January 2000a. ISSN 0003-066X. URL http://www.ncbi.nlm.nih.gov/pubmed/11392867.

Richard M. Ryan and Edward Deci. *Intrinsic Motivation and Self-Determination in Human Behavior*. Plenum Press, 1985. ISBN 0-306-42022-8.

Richard M. Ryan and Edward L. Deci. *Handbook of self-determination research*. University Rochester Press, 2002.

Richard M Ryan and Jerome Stiller. The social contexts of internalization: Parent and teacher influences on autonomy, motivation, and learning. *Advances in motivation and achievement*, 7:115–149, 1991.

Richard M. Ryan, C. Scott Rigby, and Andrew Przybylski. The Motivational Pull of Video Games: A Self-Determination Theory Approach. *Motivation and Emotion*, 30(4):344–360, November 2006. ISSN 0146-7239. doi: 10.1007/s11031-006-9051-8. URL http://www.springerlink.com/index/10.1007/s11031-006-9051-8.

Rm Ryan and El Deci. Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary educational psychology*, 25 (1):54–67, January 2000b. ISSN 0361-476X. doi: 10.1006/ceps.1999.1020. URL http://www.ncbi.nlm.nih.gov/pubmed/10620381.

Maung K Sein, Ola Henfridsson, and Matti Rossi. RESEARCH ESSAY ACTION DESIGN RESEARCH. 35(1):37–56, 2011.

Ping Zhang. motivational affordances : Reasons for ict design and use. *Communications of the ACM*, 51(11):145–147, 2008.